Final Project - Introduction to Cryptocurrencies (67513)
Maximizing Revenue in The Lightning Network

Alon Netser	Daniel Afrimi	Ariel Elnekave
311602536	203865837	308413715

August 31, 2020

1 Introduction

The Lightning Network [1] has been suggested as a solution to Bitcoin's long-known scalability issues [2, 3, 4, 5, 6]. The Lightning Network promises to improve both the number of transactions that can be processed, and the latency per transaction. The Lightning Network, along with other payment channel networks, aims to move the majority of transactions off-chain, with the guarantee that the state of channels can be committed back to the blockchain at any time, in a trustless fashion. It thus solves the problem of a limited transaction rate by limiting the communication on payments to a much smaller set of participants, and lowers the number of interactions with the blockchain.

An important feature of payment channel networks is that they also support transactions between participants without a direct channel between them, using *multi-hop routing* [1]. In order to incentivize participating in other parties transactions, intermediate nodes may require fees for transferring the money forwards to the next node in the route.

In this work we analyze the potential revenue in creating channels in the Lightning Network. The amount of money we lock in channels is treated as an *investment*. Our cost of investing in the Lightning Network includes both the locked money in the channels, as well as the fees to pay the miners to include the channel's creation transaction in the blockchain. Indeed, traditional investments (e.g. stocks, real-estate) are made by locking money in order to maximize the yield and minimize the risk, and there are also side payments such as fees to mediators, so our methodology falls into this framework.

We analyze different policies (a.k.a. *agents*) which all aim to maximize the profit gained from the fees. The main challenge is deciding which channels to create and how much money to lock in them, in order to be attractive for other parties to route transactions through them. We further discuss different trade-offs faced by the agents and analyze them using simulations.

The code we used is publicly available in our GitHub repository:
github.com/AlonNT/lightning-project.

2 Previous Work

A great deal of the work studying the lightning network is concerning security vulnerabilities and possible attacks. Tochner et al. [8] examine an attack in which malicious nodes join the network and establish new channels in strategic locations to maximize the number of routes that go through. This forms a denial of service attack on the network, as these malicious nodes can not respond to their incoming requests to forward transactions. This is very relevant to our work because we try to use similar methods - form new channels to affect the routing in the network. However, we do that to another end: Maximize the revenue from transaction fees, instead of maximizing the amount of routes passing through us.

There are other works which add new channels to the network in order to form some kind of attack. Mizrahi et al. [9] and Harris et al. [10] both use a somewhat similar way of overloading network's channels with lots of small transactions. [9] shows how the bound on the number of concurrent transactions going through a single channel can be exploited to form a cheap denial-of-service attack by choking channels with lots of tiny transactions. The attack in [10] goes further and show that stealing money is possible, using a dishonest one-sided closure of the channel. The other party can appeal to claim his money back, and this appeal needs to be published to the blockchain. If the blockchain is overloaded with pending transactions this appeal will not make it in time and the attack will succeed.

As far as we know, no significant work was done to study the potential profit from the fees in the Lightning Network. In [11] the author discusses the reason for the fees and why they are important for the incentives of the participants, but it does not show concrete methodologies for maximizing the revenue from the fees. It also states that one of lightning's prominent application developers, Alex Bosworth, reported a poor monthly income of roughly 2$. However, this developer did not necessarily established channels in order to maximize his gain from the fees, so we can hopefully do better.

3 Problem Setting

3.1 Resemblance to Reinforcement Learning

The setting can resemble the setting of Reinforcement Learning - an agent operating in an environment which is the Lightning Network. The agent observes a state s and decides to perform some action a which results in a new state s' and a reward r. The agent is not aware of the distribution from which the new state and reward are generated (given its previous state and action). There are several aspects for the state representation in our task. Generally speaking, the state should include the structure of the multi-graph describing the Lightning Network, which includes the connections between the nodes and the description of each channel.

Unfortunately, getting data for this problem is hard due to privacy reasons. For example, the balances of each channel are private. The transactions themselves are also unknown, because they are performed off-chain between the sender and the receiver without being published to the blockchain. Furthermore, the intermediate nodes on the route of a transaction are not aware of the full route, due to the "Onion Routing" mechanism (they are only

familiar with their local input and output) [7].

In real life, the distribution of the transactions is unknown to the agent. This includes which two parties will participate in the next transaction, and how much money will be transferred. In order to fully simulate the environment one needs to connect to the Lightning Network with an actual node and monitor its income from the fees. This way one can actually model the problem using RL, and try to use different RL algorithms in order to maximize his reward. We leave it for future work, as this is costly.

3.2 Our Setting - A Sub-Graph of The Lightning Network

As most of the data for the problem is private and we can't observe it, we decided to model the problem in a different way than RL. The environment itself is now known to the agent, and we are left with an optimization problem - given the Lightning Network's graph (at some time-step) and some fixed distribution over transactions, maximize the reward received from the fees. We model the problem as a simulator and an agent communicating between them.

We took a dump of the Lightning Network from May 2020 and used it for our experiment. As this is a large graph (about 5,000 nodes and 30,000 channels), and the algorithms we use are quite heavy, we sub-sample the graph by sorting the nodes by their total capacity (see subsection 4.2) and taking a sub-graph containing the nodes $j, j+1, ..., j+(N-1)$ for $N = 50, j = 50$.

Note that in real life the graph itself changes constantly due to nodes joining/leaving the network, as well as nodes which change their connections during time. In order to simplify the model we do not take this into account - in our experiments the connections are constant. However, the channels' balances change constantly due to the transactions being performed by the simulator.

One can argue that the most connected (highest total capacity) nodes are the most active ones. After all, locking money is costly and it would be inefficient to have lots of high capacity channels for an idle node. Modeling the Lightning Network traffic with the N most prominent nodes can be therefore interpreted as modeling the de-facto part of the network, i.e the active part of it. Another way to look at it is as a predefined strategy of the agents - instead of looking on all of the nodes, which may be isolated or very inactive, the agents focus on the most important nodes in the network which has higher traffic and therefore more fees can be gained.

3.3 Handling Missing (Private) Data in The Lightning Network

As mentioned previously, some information in the Lightning Network is missing due to privacy reasons, and compromises had to be made in order to enable simulating our experiments.

3.3.1 Channel's Balances

In real-life, each channel's capacity is distributed in some way between the two parties of the channel. This is an important aspect, as if all of the channel's capacity is at one side of the channel, transfers are possible only in one direction. Furthermore, if one side of the channel

has positive balance but not enough for transferring the required amount, the transaction is revoked.

For our simulator to work we must have some balances for each channel, as each transaction made in the simulator change them. As the balances of each channel are unknown, we chose them in the following naive way - each channel's capacity is divided between its two parties uniformly at random as a fraction between 0 and 1. This seems like a fair compromise, as we didn't have the ground-truth and we wanted our simulator to resemble the real network where some channels are imbalanced.

3.3.2 Node's Specific Implementation of The Lightning Network

The are three implementations to the Lightning Network - LND, CLightning and Eclair. Each implementation determines the routing algorithm which slightly differ between the three. Each implementation also determines the default established channel's policy (how much fees to demand, etc). As Mizrahi et al. [9] showed, the vast majority of the nodes in the Lightning Network use the LND implementation (about 90%) so we chose to focus on this implementation alone.

Note that the only aspect of the implementation affecting our simulations is the routing algorithm each node uses to route its multi-hop transaction. Focusing on LND implementation seems like a fair compromise, since if we show we can benefit from the fees assuming LND implementation it'll also be beneficial in real-life, since almost all of the nodes in the network use this implementation.

3.3.3 The Distribution Over Transactions

Our simulator uses the uniform distribution on the nodes in the graph to pick the two nodes in the next transaction. The amount of money being transferred is fixed. This is of course not optimal, but seems like a fair compromise, as the real life data is private and we do not know the distribution over the amount of money in each transaction, as well as for the distribution over the nodes participating in transactions.

3.4 Establishing New Channels

When two nodes want to establish a new channel in the network they need to agree on the initial balances of the channel and the fee policies of each side. In our experiments we ignore this issue - when an agent chooses a node to connect with it always gets its permission. When our agent establishes a channel, the fee policy of the other node is defined as the default one (as defined in the LND implementation of the Lightning Network).

Regarding the balances of the new established channel - the agent chooses an amount of money x to lock in the channel, and the channel is established with balance x in each side (to the channel's capacity is $2x$). This is a compromise we had to make in our simulations, because if the agent will always establish channels with all the balance in his side, he'll never be able to receive money as he won't be able to participate in other's transactions. One can argue that this compromise make sense, as the agent can decide to buy some goods from the other party of the channel, this way achieving the required balance.

4 Methods

4.1 Random Agent

This one is the simplest algorithm, used mainly as a baseline for other more sophisticated ones. Given the input parameter d, the random agent opens d channels with d nodes selected uniformly at random from the graph, where its initial funds are evenly divided between the channels. The fee policy of each channel it created is the default policy.

4.2 Greedy

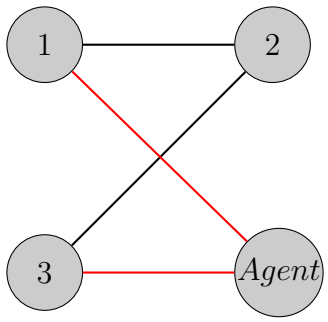
We defined three methods for scoring the nodes, each score define a corresponding greedy algorithm. The greedy algorithm orders the nodes in a descending/ascending order according to their score, and then choose the first d nodes (where d is an input parameter) and establish channels with them dividing its funds equally between the channels. The fee policy of each channel it creates is the default policy.
The scoring methods are the following:

1. **Channel capacity**: Each node's score is its total capacity - the sum of the capacities in all of the channels it's participating in.

2. **Graph-Degree**: Each node's score is its degree in the multi-graph.

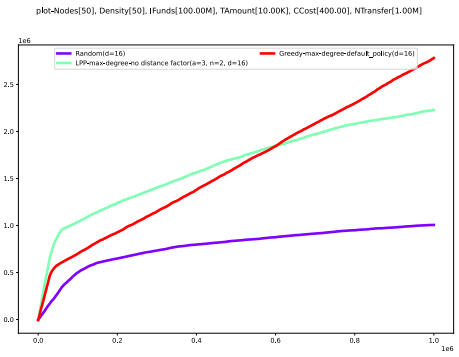
3. **Routeness**: Each node's score is the number of routes it might participate in, when some two nodes in the graph will make a transaction.

   For example, suppose the pair of channels $\{1, 2\}, \{2, 3\}$ is highly ranked and the agent want to enable bypassing it through his node. The nodes 1 and 3 will get high scores and the agent will open channels with them. Then, every route that has a sub-route $1 \to 2 \to 3$ (or $3 \to 2 \to 1$) can now use the alternative sub-route and the pass through the agent. The following graph demonstrate this method.



4.3 Lightning++

The motivation for this algorithm is taken from kmeans++ [12] clustering algorithm, and this is where it got its name. In the traditional kmeans clustering algorithm the initial centroids are chosen uniformly at random. In kmeans++ the initial centroids are sampled according to a distribution which gives high probability to nodes that are distant from the



(a) The greedy agent maximizing the degree reachs the highest revenue.

Figure 1: Performance of the agents, using the best hyper-parameters for each one.

previously selected centroids. This enables choosing the initial centroids in a random way which results in nodes that are far from each other (which helps the algorithm to cluster better), ignoring outliers which are a few data-points that are extremely far from the rest.

We wanted to add randomness to our agents policies, so instead of selecting greedily the best node (according to some ordering), we define a distribution over the nodes where each node probability is according to its score - higher score implies higher probability to select this node. Eventually the Lightning++ agent works similar to the greedy agent, meaning that it selects nodes with high rank according to some score function, but it does so in a random way so not always the "best" node will be selected.

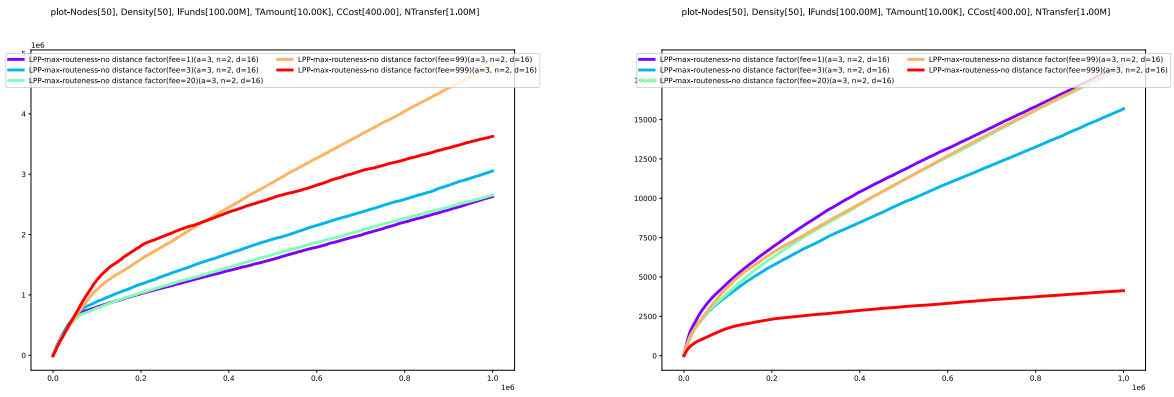
5 Results

The algorithms we use are the random agent as a baseline, and the greedy and Lightning++ with different scoring methodologies. All agents are given the same initial funds they can allocate for new channels. The experiments are repeated 5 times, and the plots are the average of the revenue in each time-step between all 5 experiments.

Figure 1 shows the results of the experiment where each agent is given with its best hyper-parameters. The figure shows that the greedy agent with graph degree method wins, then L++ agent with graph-degree method and at last the random agent.

5.1 Fees Tradeoff

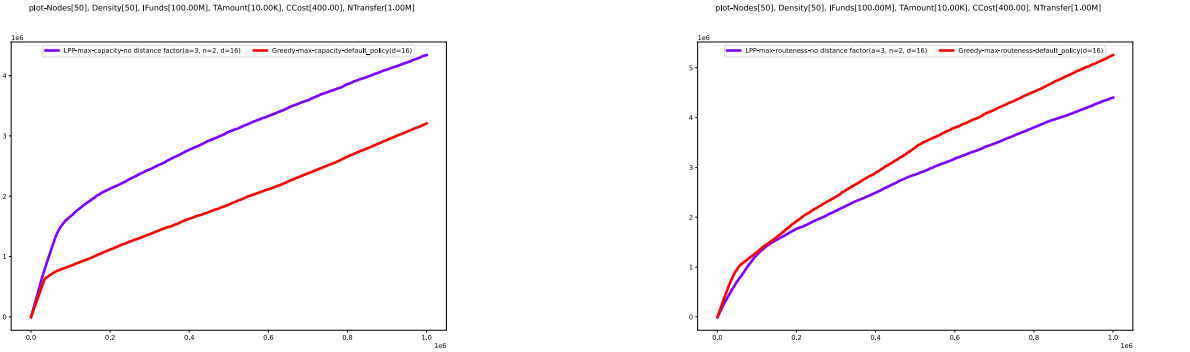
The agents are faced with a tradeoff deciding how much fee to require - high fees means more profit per transaction, low fees means more transactions passing through. Figure 2 shows the performance of the L++ agents when using different values for the fees, showing that the best base fee for routeness method is 99 mili-satoshis. Note that when the fee is low the number of transactions is high, but the revenue is not necessarily higher. We examined other agents as well and the results are similar.



(a) Revenue

(b) Number of transactions

Figure 2: The performance of the L++ agents when using different values for the fees.



(a) Greedy v.s L++ in Channel Capacity method

(b) Greedy v.s L++ in Channel Routeness method

Figure 3: Greedy v.s L++

5.2 Greedy Algorithm v.s. Randomized Algorithm

We tested the greedy agent versus the Lightning++ agent. This main question we ask is whether randomness helps the performance of the policies. Figure 3 shows the performance of the agents L++ and greedy, showing that in channel capacity method L++ beats, but in routeness method greedy wins. So the answer to the questions whether the randomness helps policy is not conclusive.

5.3 How Good is the Default Policy?

We tested the default policy versus a custom policy we created - the minimal values for the channels you wish to "replace". Figure 4 shows the performance of the greedy agent in the two modes, showing that when we used the default policy the results were better. We examined other hyper-parameters as well and the results were similar.

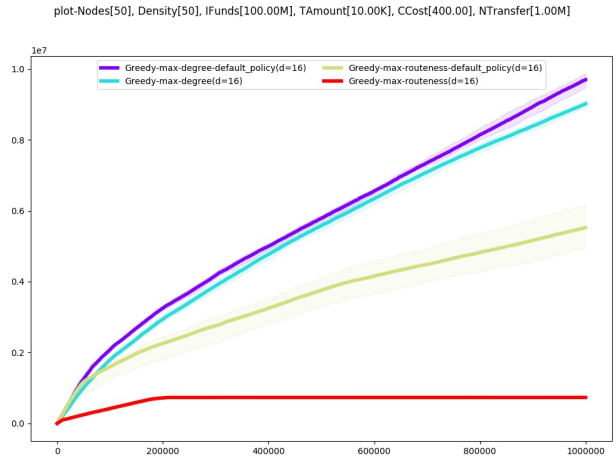


Figure 4: Greedy agent with default policy v.s. custom policy

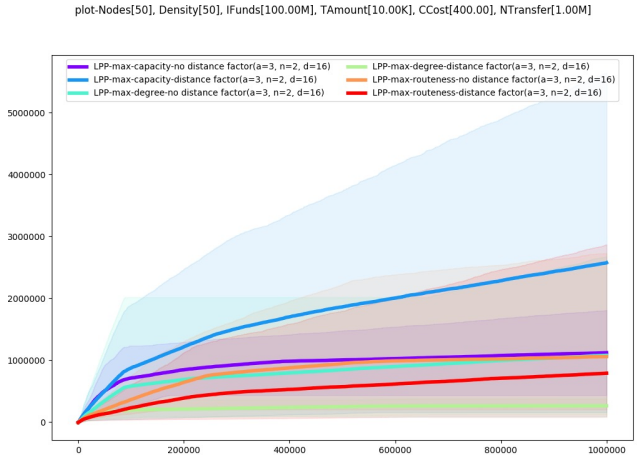


Figure 5: Lightning++ agent with and without distance consideration

5.4 Are Distant Nodes Preferable?

We tested the Lightning++ agent in two modes - one mode selected the best nodes according to some score-function, and the other mode include a distance parameter as well, meaning preferring distant nodes, in addition to the optimized score. Figure 5 shows the performance of the Lightning++ Agent, showing that the results were not conclusive - in some methods like routeness and degree using no distance parameter was better, although in capacity method the results were different.

6 Conclusion and Future Work

In this work we showed initial results regarding the benefit one can gain from the fees in the Lightning Network. We examined different algorithms which all try to establish channels in order to maximize their revenue from the fees. This can be treated as an investment, where the investor locks money in channels of the network and gain revenue from the fees he collects when others transfer money though his channels.

One direction for future work is to use sophisticated ways to extract (some of) the private in the Lightning Network, such as the balances and the transactions distribution. This way an environment for reinforcement learning can be created, and finally be examined in real life.

Another direction for future work is to come up with more sophisticated algorithms that get higher profit that the methods we described. When the results will be satisfying enough it'll be interesting to compare the potential gain using the fees and other types of investments, specifically in the area of Bitcoin (e.g. mining).

References

[1] Joseph Poon and Thaddeus Dryja. 2016. The bitcoin lightning network: Scalable off-chain instant payments.

[2] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. 2018. Deconstructing the blockchain to approach physical limits. arXiv preprint arXiv:1810.08092 (2018).

[3] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. 2016. On scaling decentralized blockchains. In International conference on financial cryptography and data security. Springer, 106–125.

[4] Christian Decker and Roger Wattenhofer. 2013. Information propagation in the bitcoin network. In IEEE P2P 2013 Proceedings. IEEE, 1–10.

[5] Yonatan Sompolinsky and Aviv Zohar. 2015. Secure high-rate transaction processing in bitcoin. In International Conference on Financial Cryptography and Data Security. Springer, 507–527.

[6] Aviv Zohar. 2017. Securing and scaling cryptocurrencies.. In IJCAI. 5161–5165.

[7] M.G. Reed, P.F. Syverson and D.M. Goldschlag. 1998. Anonymous connections and onion routing.

[8] Saar Tochner, Stefan Schmid and Aviv Zohar. 2019. Hijacking Routes in Payment Channel Networks: A Predictability Tradeoff

[9] Ayelet Mizrahi, Aviv Zohar. 2020. Congestion Attacks in Payment Channel Networks

[10] Jona Harris, Aviv Zohar. 2020. Flood & Loot: A Systemic Attack On The Lightning Network

[11] You Can Now Get Paid (a Little) for Using Bitcoin's Lightning Network, by Alyssa Hertig, Coindesk. https://www.coindesk.com/you-can-now-get-paid-a-little-for-using-bitcoins-lightning-network

[12] Arthur, David, and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Stanford, 2006.