Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Designing a Dynamic Micropayment Channel Network

Bachelor Thesis

Corsin Gutkowski

gucorsin@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Conrad Burchert
Prof. Dr. Roger Wattenhofer

December 14, 2017

Acknowledgements

I thank Conrad Burchert, who gave me great advice on many questions throughout this thesis, and who was always passionate about the topics of our long and profound meetings.

Abstract

In this thesis, we design a micropayment channel network from scratch. The resulting network should be convenient for most participants and not for few. We propose a strategy to connect participants in a micropayment channel network and set fees on channels accordingly.

In the first part, a model for a micropayment channel network is defined, based on which we develop a strategy later on. The model assumes infinite capacities on micropayment channels. A key factor for the strategy design relies on the transaction flow inside the network. For network creation and updates, corresponding algorithms are constructed which are based on mathematical properties the network should fulfill. Algorithms for network updates are iteratively applied to the network.

The strategy, when applied by participants, converges to a stable network state in which fee update differences are negligible.

Contents

Chapter 1

Introduction

Nowadays we almost daily receive news from the world of cryptocurrencies. Especially Bitcoin, whose concept was introduced in 2008 by Satoshi Nakamoto [2], makes headlines of late. Undoubtedly, cryptocurrencies and their underlying blockchain technology have to be considered an alternative - or rather addition, to our financial system.
However, Bitcoin is by far not perfect and still has major challenges to master, one of them being scalability. With the current block size capped at 1MB and new blocks being found on average every 10 minutes only 7 transactions/s are expected, considering an average transaction size of 250 Bytes. Thus, more and more transactions get added to backlog and need even longer to be confirmed as the already recommended waiting time of several blocks to avoid double spending. In comparison, credit card company Visa claims to support 56'000 transactions/s.[1] Latest adoptions of the SegWit protocol change and block limits up to 8MB with Bitcoin Cash[2] are only partial solutions.

Micropayment channel networks [1, 8] operating on top of the blockchain are one solution to solve Bitcoin's scalability problem. The idea behind a micropayment channel is that almost limitless transactions between two channel parties can be handled with smart contracts off-blockchain. Thus, giving space to process other transactions on the blockchain and eventually increasing the transaction volume for a broader user base. Micropayment channels can be connected to a larger network where any two participants, who do not have a common channel, can send transactions over larger distances to each other. A small fee has to be paid for every hop in between sender and receiver. Compared to the fee charged for sending a transaction via the blockchain, fees on a micropayment channel are expected to be much smaller. In chapter 2 the concept of micropayment channels is going to be explained in more detail.

[1] https://usa.visa.com/dam/VCOM/download/corporate/media/visa-fact-sheet-Jun2015.pdf
[2] https://www.bitcoincash.org

1.1 Motivation and Goals

How participants of a micropayment channel network connect among themselves and whether the resulting network structure still preserves one of Bitcoins core principles, decentralization, has to be questioned [4]. Critics believe that large institutions, like banks, could dominate micropayment channel networks by being centralized hubs which route transactions between participants. Only wealthy participants could afford to create enough channels to maintain such a routing service making them eventually even wealthier. We are not going to judge if this is a good or a bad thing. We instead would like to design a micropayment channel network from scratch.

Our network should be convenient for most participants and not for few. We believe that the *fees* on micropayment channels are an essential criterion for such a network. On the one hand, participants want to minimize their fee payments on transactions by choosing profitable routes, on which fees are presumably low. On the other hand, participants want high fees on their own channels, paying out fee earnings by forwarding transactions. Either way, fees play an important role, and ultimately the question arises, *how* should we choose fees on micropayment channels?

In this thesis, we propose a strategy on how to connect participants in a micropayment channel network and set fees on channels accordingly. The strategy should achieve a globally stable network state in which micropayment channel fees are persistent. Ideally, the resulting network structure would be decentralized.

Chapter 2

Background

Descriptions and illustrated figures throughout this particular chapter are inspired by Burchert et al. [9]. Micropayment channels are first introduced by Hearn and Spilman [6] and later micropayment channel networks have been proposed simultaneously by Decker et al. as Duplex Micropayment Channels [8] and by Dryja et al. as the Lightning Network [1].

2.1 Micropayment Channels

A micropayment channel is set up between two participants of a micropayment channel network who want to exchange currency without broadcasting transactions to the blockchain, apart from a one-time funding transaction to open and a final commitment transaction to close the channel.

To create a channel an m-of-n multisig transaction output can be used, which is only spendable if m private keys of the corresponding n public keys of the output are provided. In case of micropayment channels, a 2-of-2 output is created including both channel parties.

Opening a channel consists of two transactions: a funding transaction, broadcast to the blockchain, and a commitment transaction stored by both channel parties.
In a funding transaction, parties declare how much funds they want to put in a channel. Thereby, the channel balance is determined. Once funds are locked-in, they are only usable inside a channel and are not transferable to other channels. Inputs to the funding transaction usually reference previously unspent singlesig outputs fulfilling Proof of Ownership.
Furthermore, a commitment transaction is created which, when being appended to the blockchain, outputs the balance to the channel parties respectively. The commitment transaction is a security that guarantees locked-in funds are returnable at any time after channel creation. In Figure 2.1 channel creation is depicted.

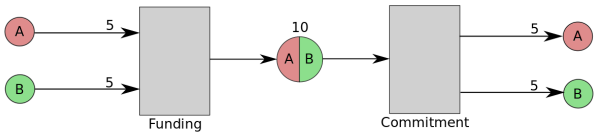

Figure 2.1: Creation of a micropayment channel. Boxes represent transactions and circles inputs or outputs. The multisig output/input in the middle is only spendable if both channel parties sign the commitment transaction.

A channel can be updated by replacing the commitment transaction with a newly agreed on commitment transaction representing the new channel state. Figure 2.2 shows a channel update. With every update, the balance on the channel gets shifted. Updates are the equivalent of sending transactions between channel parties. As both private signatures are needed to create a new commitment transaction locked-in funds are safe at any time. Eventually, a channel gets lopsided and currency must flow in the other direction to balance the channel, or the channel has to be closed and reopened via the blockchain with a new balance.

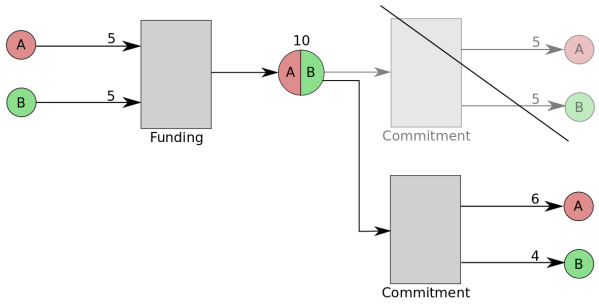

Figure 2.2: Update of a micropayment channel. A new commitment transaction is created and signed by both channel parties, representing the new channel state where A will receive one unit more, and B one unit less than in the old state.

Both parties can close the channel at any time by publishing the last agreed on commitment transaction. To prevent parties from broadcasting old commitment transactions, and hence, violating the current channel state, more sophisticated smart contracts, like timelocks [8] or revocable transactions [1], can be used.

Once a channel is established new transactions can be created within seconds depending on the latency and throughput of the channel parties. In comparison to Bitcoin's blockchain, which needs several minutes to hours depending on the traffic to confirm a transaction, this is a significant improvement.

2.1.1 Timelocks and Revocable Transactions

Duplex Micropayment Channels [8] make use of timelocks for replacing commitment transactions. The concept relies on using the lock_time field of a transaction, meaning a transaction will only be accepted in the blockchain after time T has elapsed. Replacing a transaction will decrement the timelock of the new transaction. For example, when a first transaction is created with a timelock of T=10 the second transaction replacing the first one will have T=9 and so forth, see Figure 2.3. There is always a clear order amongst all transactions.

When the timelock of the latest transaction has elapsed the channel should be closed, otherwise older transactions might also get publishable, which would violate the most recent agreed-on channel state. An invalidation tree described in [8] can be used to prolong the lifetime of channels.

In the Lightning Network [1] revocable transactions are used. Is one party not behaving correctly and eventually tries to cheat, its counterparty can claim the whole channel balance. For this purpose, each channel party creates a secret for each transaction. After a transaction gets invalidated by a new one, secrets of the old transaction get exchanged. A personal commitment transaction is created for both channel parties, which is already signed by the counterparty and can be published with the party's own signature at any time. The set up of a revocable transaction is illustrated in Figure 2.4. The scenario in which one party can punish the counterparty is possible when old transactions are broadcasted and secrets are known beforehand. Every secret has to be stored, and thus, the construction of revocable transactions needs additional storage space.

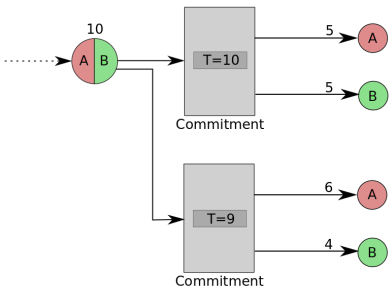

Figure 2.3: Transaction replacement with timelocks. For consequent transactions the lock_time T is decreased.

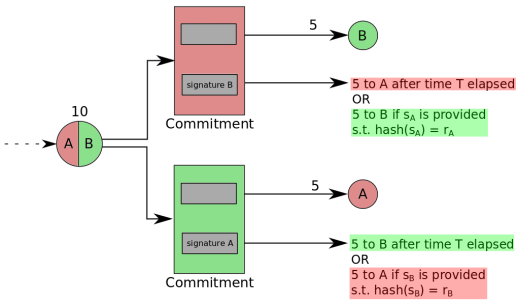

Figure 2.4: Revocable transaction. The coloring indicates to which channel party the transaction/-output belongs.

In general, participants can interact trustlessly with other participants in the network using one of the presented constructions, as long as channels are actively maintained. By knowing that one channel party is absent, or in case of

a successful Denial of Service attack, the channel can be abused as both concepts rely on sufficient time to counteract misbehavior.

2.1.2 Hashed Timelocked Contracts

A micropayment channel network is reasonable when participants do not need to be connected to everyone else. Thus, different channels can be connected to send transactions over multiple hops. E.g., participant A wanting to send a transaction to C over B implies that A and B as well as B and C maintain a channel ch_{AB} and ch_{BC} respectively, see Figure 2.6. In this scenario, B would be an intermediate hop.

To construct transactions Hashed Timelocked Contracts (HTLCs) are used which guarantee atomic currency exchange over multiple channels. Figure 2.5 illustrates the setup.

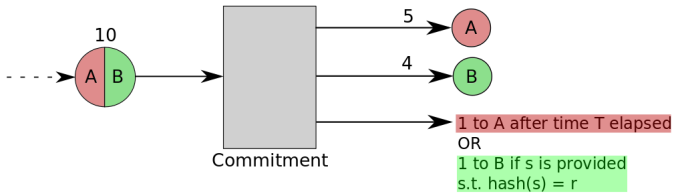

Figure 2.5: HTLC commitment transaction. There are three different outputs, where the first two return directly the balance to A and B respectively. The third output can be claimed by A after time T has elapsed, or by B if B can provide a secret s s.t. $hash(s) = r$. The underlying transaction goes from A to B.

The third conditional output guarantees that in case of a non-cooperating intermediate node all channel parties receive their rightful share after a certain time. Therefore, the timelock T has to be decreased for each consecutive HTLC, otherwise, currency loss might occur. Additionally, each party is given enough time to pull their funds by providing the secret to their HTLC.
A transaction of one unit from A to C with HTLCs could look as shown in Figure 2.6.

HTLCs can be established in any chain of any length consisting of different payment channels. As an incentive for intermediate hops to forward transactions a small fee is charged for using the service of the channel, i.e., A would propose in Figure 2.6 slightly more than one unit to B. Fee payments are also justified as the balance on a channel gets shifted, which is only beneficial for balancing a lopsided channel. Channel parties decide in collaboration on a channel fee.
After a successful transaction with HTLCs, channel parties do not need to broad-

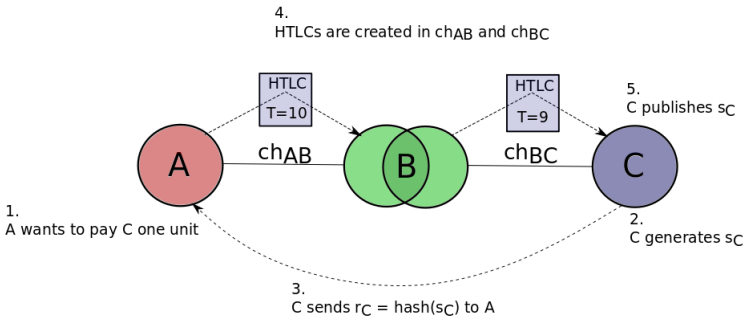

Figure 2.6: Atomic exchange over two micropayment channels with HTLCs. As soon as HTLCs are successfully established within each channel, C can publish secret s_C and pull the unit from B, and B from A consecutively.

cast their contract and can just replace their HTLC with a new commitment transaction without an HTLC. HTLCs can be combined with timelocks or revocable transactions changing the output of the HTLC accordingly.

Chapter 3

Design Strategy

We start by formally defining an appropriate model for a micropayment channel network. Later on, we propose properties which a network should feature when being created, and when being updated. A key factor for our strategy design will rely on the transaction flow inside the network.

3.1 Model

In the following, we present assumptions and definitions for different components of a micropayment channel network.

Definition 3.1 (Capacity)**.** The *capacity* of a micropayment channel is the sum of the channel balance.

Assumption 3.2. Micropayment channels have infinite capacities. Transactions can be sent regardless of capacities.

Definition 3.3 (Network Graph)**.** A *network graph* $G = (V, E)$ consists of n different participants. Each participant is represented by a unique node $v \in V$. Two participants X, Y having an open micropayment channel ch_{XY} are connected by an edge $e \in E$. It holds that $|V| = n$, and $n-1 \leq |E| \leq \frac{n(n-1)}{2}$. Edges in a network graph are undirected.

Assumption 3.4. A valid network graph is connected.

Apart from the previously stated assumptions, micropayment channels, represented by edges in our model, and resulting transactions are constructed as described in Chapter 2, e.g., transactions can be secured with HTLCs on the sending route.

Definition 3.5 (Connection Rule)**.** A *connection rule* is a condition under which two nodes should connect in a network graph.

A simple example for a connection rule $R1$ could be: *"connect two nodes s, r if s wants to send a transaction to r"*, see Figure 3.1.
A more complex example could be: *"connect two nodes u, v iff conditions $c_1, .., c_k$ are met"*.
Each application of a connection rule will result in a new network graph G' with $G' = (V, E \cup \{e\})$, where $e = (u, v)$ and u, v are the involved nodes. Similarly, a disconnection rule can be defined resulting in $G' = (V, E \setminus \{e\})$.

If a network graph becomes disconnected, e.g., when no transactions are being sent between two different groups of participants, we can treat each subgraph as an independent network graph. Thus, we can also handle such cases even though we consider only connected graphs by Assumption 3.4.

Definition 3.6 (Honest Actor)**.** An *honest actor* actively participates in a network and follows the proposed strategy.

Assumption 3.7. Participants in our model behave as honest actors.

The basic network structure is settled. Now, we need to define transactions between participants. We quantify transactions by their volume (concerning the transaction count) being trafficked during a certain period. Such a quantification assumes some knowledge about the future to be useful, e.g., how many transactions are going to be sent during a certain period from node s to node r. The more accurate the prediction is, the better participants can adapt the network by making advantageous connections beforehand.

Definition 3.8 (Transaction Matrix)**.** A *transaction matrix* of a network with n participants is defined as $T_m \in \mathbb{N}^{n \times n}$. Matrix entry $T_m(s, r)$ denotes the number of transaction being sent during a certain period, where s is the sender and r the receiver of the transactions. It holds that $\forall s \in \{1...n\} : T_m(s, s) = 0$.

For example, matrix entry $T_m(s, r) = 5$ would be read as: *"Node s sends 5 transactions to Node r"*. See Figure 3.1 for an example.
The observed period in which these transactions are being sent is variable for each transaction matrix. It is even possible that each entry of the transaction matrix represents a different period. This difference depends on how long we expect a channel to persist.
Throughout this thesis, the index s stands for sender and the index r for receiver.

Definition 3.9 (Fee Matrix)**.** A *fee matrix* of a network with n participants is defined as $F_m \in \mathbb{R}+^{n \times n}$. Matrix entry $F_m(i, j)$ denotes the amount of fee to pay when routing a transaction over edge $e = (i, j)$. It holds that $F_m(i, j) = F_m(j, i)$, and further $\forall i \in \{1...n\} : F_m(i, i) = \infty$.

Matrix entry $F_m(i, j) = \infty$ denotes that node i and j are not directly connected, i.e., they have no common payment channel. A fee matrix is similar to an adjacency matrix. Note, fees are positive.

$$\begin{pmatrix} 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 4 & 3 & 1 \\ 0 & 4 & 0 & 2 & 5 \\ 2 & 0 & 1 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

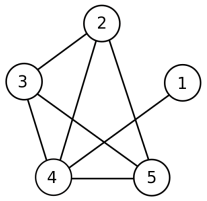

Figure 3.1: Transaction matrix and the resulting network graph, created under the connection rule $R1$: *"connect two nodes s, r if s wants to send a transaction to r"*. Note, node 5 is only a receiver node as the 5-*th* row of the matrix contains only zero entries.

To access the fee on an edge the weight function:

$$f : E \to \mathbb{R} \tag{3.1}$$

is defined, which takes as an input edge $e \in E$ and returns the associated fee, see Figure 3.2. The weight function matches each edge with a possible value if that edge should exist.

$$\begin{pmatrix} \infty & \infty & \infty & 4 & \infty \\ \infty & \infty & 3 & 1 & 1 \\ \infty & 3 & \infty & 2 & 5 \\ 4 & 1 & 2 & \infty & 1 \\ \infty & 1 & 5 & 1 & \infty \end{pmatrix}$$

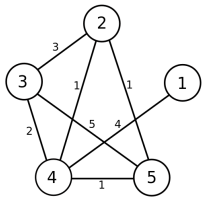

Figure 3.2: Network graph from Figure 3.1 with labeled fees on the edges according to the fee matrix on the left. Here, the highest fee would be paid on edge $e = (3, 5)$, namely $f(e) = 5$.

Participants only have a partial view of the complete fee matrix as they are interested in the fees paid along routes which route over their channels. The same holds for the transaction matrix, where participants are interested in the traffic on their channels. Participants can use routing protocols, which are not subject to this thesis, to obtain information on different routes. By sending transactions, participants gradually get a local picture of the transaction flow, which in the end allows predictions about the transaction matrix.

For clarity, the following expressions are equivalent:

- *node*, *participant* and *channel party*
- *edge* and *channel*

3.2 Fee Strategy

When we want to develop a strategy, we first need to consider what properties are important for a micropayment channel. The strategy is defined from a global perspective but is locally applied by participants.

One important property is the connectivity of the participants. Participants should only connect to *few* other participants. Firstly, because locked-in funds will get proportionally smaller and smaller with each created channel (as in reality participants have only limited funds to spend). Thereby, channels become more useless to route a transaction over. And secondly, opening and closing a lot of channels will burden the blockchain with more transactions. Thus, we want the network to penalize creating too many channels.

By another property we want to capture that participants act rationally to some extent, i.e., they *care* about the fee being paid on a channel, either as customer or owner of the channel. When offering a channel to other participants, routing a payment over that channel should be attractive. For example, a channel with the fee close to the blockchain transaction fee (on-chain cost) will most likely not be picked by transactions which need more than one hop to their destination. Therefore, we also want participants to avoid large fees, which indirectly suggests choosing *good* channels.

We propose the following loss function which combines both previously mentioned properties:

$$l(G, f, T_m) = \sum_{(s,r)} T_m(s,r) \sum_{e \in SP(s,r)} f(e) + B|\{e \in E | \exists s, r : e \in SP(s,r)\}| \quad (3.2)$$

The loss function $l : ((V,E), f, \mathbb{N}^{n \times n}) \to \mathbb{R}$ takes as arguments a network graph $G = (V,E)$, weight function f, transaction matrix $T_m \in \mathbb{N}^{n \times n}$ and calculates the resulting loss. The transaction matrix T_m can be arbitrary as long as the resulting network graph G is connected, i.e., it exists a traffic flow which includes all participants. The weight function is dependent on the graph and its underlying fee matrix, where participant decide on channel fees.
The double sum of the loss function captures all the fees being paid by all transactions. On the right side of the plus sign are the costs for creating all channels, i.e., all edges in the network graph belonging to some shortest path (SP). The shortest path is the cheapest route from sender to receiver with respect to fees being paid on the route, not hops concerning the distance of the route.

The goal now is to *construct* a network graph for which the loss, given a transaction matrix, is minimal. I.e., we want to provide a network graph for which we can reason *why* existing edges are formed, and also, *why* these edges are favorable to minimize the loss potentially.

Trivial solutions should be prevented, like creating a graph with no edges, or a minimally connected graph with 0 fees being paid on the edges, for which the loss would be truly minimized. Therefore, the weight function f should be of the form $f : E \to \mathbb{R}+ \setminus \{0\}$, which is also in favor of network participants who want to make some profit eventually. Additionally, we constrain the right-hand side by assuming that the shortest path must exist for each sender and receiver of a transaction.

We define the following cost function $c : (L, f) \to \mathbb{R}$ to access the costs on a list L of connected edges:

$$c(L, f) = \sum_{e \in L} f(e) \tag{3.3}$$

The function sums up of all fees along the edges of the list. An example of such a list could be the shortest path: $SP(s, r) = L \subseteq E$.

The shortest path can be calculated from the fee matrix $F_m \in \mathbb{R}+^{n \times n}$ and the resulting network graph $G = (V, E)$. Formally:

$$SP : ((V, E), \mathbb{R}+^{n \times n}, s, r) \to L$$

with $s, r \in V$, and $L \subseteq E$. We assume that the network graph and fee matrix are always implicitly given and just write $SP(s, r)$ to improve readability.
To calculate the shortest path we are using Dijkstra's algorithm, mainly because we do not have negative fees.

Although different sources consider negative fees plausible, e.g., for attracting more transactions (a negative fee will pay off the sender of a transaction instead of charging), we try to avoid them and only allow positive fees. The main reason behind this constraint is that allowing negative fees would possibly result in oscillating fees when trying to optimize them. For example, setting fees along a path with two edges could result in one edge fee being strongly negative $f(e^-) \ll 0$, and one strongly positive $f(e^+) \gg 0$ but the overall cost would still be zero: $c(\{e^-, e^+\}, f) \approx 0$. In the only positive fee case, the fees could also vary but not exceed the on-chain cost, or else no motivation is given to send a transaction via micropayment channel network.

3.2.1 On Network Creation

With respect to the stated loss function, we want to create a network graph for n participants. Most crucial therefore is the transaction matrix $T_m \in \mathbb{N}^{n \times n}$ from which we obtain information about the future payment flow. The fee on each edge is of less importance as fees are a given factor initially chosen by channel parties.

Algorithm 1 Network Creation

Input : T_m, B
$E = \emptyset$, $V = \emptyset$, $F_m = \infty^{n \times n}$, $TX = list()$ ▷ first list index is 1
for s, r in $1...n$ **do**
    ADD($(T_m(s,r), s, r)$) to TX ▷ tx = (#tx, sender, receiver)
SORT(TX) by #tx in desc. order
$V = \{\text{TX}[1]_{\text{sender}}, \text{TX}[1]_{\text{receiver}}\}$
$E = \{(\text{TX}[1]_{\text{sender}}, \text{TX}[1]_{\text{receiver}})\}$
DELETE(TX[1]) from TX
for all tx in TX **do**
    if either tx_{sender} **or** $tx_{receiver}$ is in V **then** ▷ connection rule
        $E = E \cup \{(tx_{sender}, tx_{receiver})\}$ ▷ $e = (tx_{sender}, tx_{receiver})$
        $V = V \cup \{tx_{sender}, tx_{receiver}\}$
        tx_{sender} and $tx_{receiver}$ decide on an initial channel fee $f_{init}(e)$
        $F_m(tx_{sender}, tx_{receiver}) = F_m(tx_{receiver}, tx_{sender}) = f_{init}(e)$
    if both tx_{sender} **and** $tx_{receiver}$ are **not** in V **then**
        swap list entry tx with next occurring unswapped list entry
        continue with next list entry
    if V contains all nodes **then**
        break for-loop
for all s in V **do**
    for all r in V **do**
        if $c(SP(s,r), f_{init}) \cdot T_m(s,r) \geqslant B$ **then** ▷ connection rule
            $E = E \cup \{(s,r)\}$ ▷ $e = (s,r)$
            s and r decide on an initial channel fee $f_{init}(e)$
            $F_m(s,r) = F_m(r,s) = f_{init}(e)$
return $G = (V, E)$, F_m, f_{init}

Algorithm 1 can be used to create a network graph for given T_m and B. The algorithm sorts tuples of the form (#tx, sender, receiver) by the highest transaction count. Consecutively, nodes and edges are added on which most transactions are being trafficked until a connected graph is achieved. For added edges, an initial fee is determined by the channel parties. To add an edge either the sender or the receiver should already be in the node set otherwise it is possible to get a disconnected graph. In the last step, edges are added where the costs for

sending transactions over the shortest path exceed the cost of creating a direct connection. Note, swapped transactions should be marked and should not be swapped back otherwise an infinite loop might occur.

Participants only consider their own transactions and corresponding receiver nodes when applying the algorithm. When creating a channel, the initial fee will always allow both channel parties u and v to send at least all their transactions to each other over their channel without exceeding the on-chain cost. The initial fee is therefore bounded by:

$$0 < f_{init}((u,v)) < \frac{B}{\max(T_m(u,v), T_m(v,u))}$$

This initialization guarantees that the channel party with more transactions pays at most on-chain cost B. The upper bound becomes smaller for a higher transaction count.

One might find the construction of Algorithm 1 counter-intuitive, especially the first part. The loss function (3.2) indicates to add all edges to the edge set E which belong to the cheapest theoretical shortest path of a fully connected graph. Figure 3.3 illustrates that our algorithm achieves a better solution, where for simplicity all fees are set to 1.

$$\begin{pmatrix} 0 & 5 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 4 & 0 & 0 \\ 0 & 5 & 0 & 5 & 0 \end{pmatrix}$$

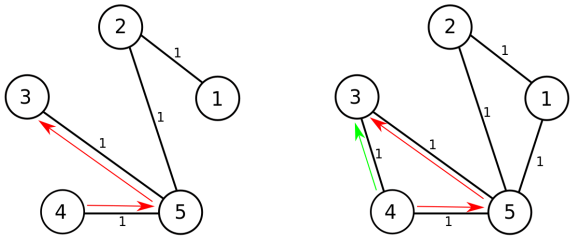

Figure 3.3: Loss comparison for different construction approaches. From left to right: transaction matrix, network graph G_1 constructed by Algorithm 1, and the network graph G_2 which uses the theoretical best shortest path. The green arrow indicates the better shortest path $SP(4,3)$ compared to the red one. We assume $B = 10$. The loss for graph G_1 is $l(G_1, f, T_m) = 36{+}4B = 76$, and for graph G_2; $l(G_2, f, T_m) = 28{+}6B = 88$.

Although Algorithm 1 achieves a promising result, the resulting network graph is not the best possible solution regarding the loss being minimal. A counterexample is shown in Figure 3.4.
Nonetheless, we believe that the construction of our algorithm is well-chosen. We can reason about *why* each edge is created. Connections are established one after another with the notion that those connections will have high traffic, and that fees on these edges are presumably low compared to other fees in the network.

Connections formed by the best solution might not be so obvious as a clear connection order might not be reasonable. In the worst case, all possible network graphs ($2^{\frac{|V|(|V|-1)}{2}}$ many) have to be considered, running in $\mathcal{O}(2^{|V|^2})$. Our algorithm runs in $\mathcal{O}(|V|^2 \log |V|^2 + |V| \log |V| + |E|) = \mathcal{O}(|V|^2 \log |V|^2)$, for sorting all transactions and finding the shortest paths.

$$
\begin{array}{c}
 \\
1 \\
\dots \\
i \\
\dots \\
n-1 \\
n
\end{array}
\begin{array}{c}
\begin{array}{cccccc} 1 & \dots & i & \dots & n-1 & n \end{array} \\
\begin{pmatrix}
0 & \dots & 0 & \dots & 0 & 0 \\
\dots & \dots & \dots & \dots & \dots & \dots \\
5 & \dots & 0 & \dots & 4 & 4 \\
\dots & \dots & \dots & \dots & \dots & \dots \\
0 & \dots & 0 & \dots & 0 & 0 \\
0 & \dots & 0 & \dots & 0 & 0
\end{pmatrix}
\end{array}
$$

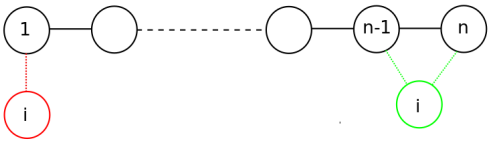

Figure 3.4: Counterexample. On the left side is the transaction matrix depicted with node i connecting to the network. The red node indicates the connection made by Algorithm 1 and the green node a potentially better solution. Connecting to node 1 will result in higher costs for sending 4 transactions to node n-1 and n each, instead of connecting to node n-1/n and send 5 transactions to node 1. The dotted line indicates a chain of nodes where fees are presumably low s.t. sending 5 transactions to node 1 from node n-1/n does not exceed the on-chain cost.

3.2.2 On Network Update

Having a strategy on how to create a network graph, we want a strategy on how to update an existing network, by updating fees and eventually creating or deleting edges.

Similarly to the network creation procedure, we also want to capture the notion of participants being rational to some extent. Hence, updating the fee on a channel should somehow be profitable for owners, in the sense that the channel continues to be attractive, *especially* for participants with a high transaction count. Here, the transaction matrix T_m is again critical, which predicts the future payment flow. To improve predictions, the Hidden Markov Model could be used which includes the past transaction flow.

We expect a real micropayment channel network to be similar to a social network, like a small-world network. Thus, we predict the transaction matrix T_m according to a small-world-like network where participants only interact with a particular group of other participants. We denote this group as *social range*.

The following procedure captures the mentioned properties for choosing a new fee, i.e., we want to obtain a new weight function f' and update the fee matrix accordingly:

$$
\begin{aligned}
f''(e) &= \min_{s\, s.t.\, \exists r: e \in SP(s,r)} \{f''(e)_s \mid *\} \\
* &\equiv \sum_r T_m(s,r) \cdot (c(SP(s,r), f) - f(e) + f''(e)_s) = B' - \gamma \\
B' &= \begin{cases} B \cdot d, & \text{if } D > SR,\ d = k \cdot (D - SR),\ 0 < k \\ B, & \text{otherwise} \end{cases} \\
f'(e) &= \begin{cases} f(e) + (f''(e) - f(e)) \cdot \eta, & \text{if } f''(e) > 0 \\ f(e) - f(e) \cdot \eta, & \text{otherwise} \end{cases} \qquad \text{with } 0 < \eta < 1
\end{aligned}
\tag{3.4}
$$

For the input edge e, a set of different fee proposals for a choice of a new fee $f'(e)$ is calculated. For each sender node s, where e belongs to the shortest path of that node, a proposal $f''(e)_s$ is determined.
The equation inside the curly brackets $(*)$ has to be resolved after the unknown, yet to be determined fee proposal $f''(e)_s$. The left side of the equation sums up all shortest paths of sender s routing over e. The actual fee $f(e)$ gets subtracted from the costs of the shortest path $c(SP(s,r), f)$ and the new possible fee $f''(e)_s$ gets added to the costs. On the right side of the equal sign is the on-chain cost B' (modified under a certain condition, which will be explained soon) diminished by a value γ. With the left-hand side being equal to the right-hand side we choose

the fee such that the costs do not exceed the modified on-chain cost B'. Thus, participants are likely to choose the channel for future transactions.

By the very nature of the equation, a possible fee proposal could be negative. Therefore, we choose either to advance the smallest proposal if it is positive, or otherwise we reduce the already existing fee. Choosing the minimum will guarantee all sender nodes whose fee proposal is bigger to continue sending transactions over the updated edge. By step-wise advancing the proposed fee, we set fees along routes more evenly and make the network more robust for fast changing transactions, i.e., fees are changing more smoothly.

In some networks, far distant nodes (in terms of hops) could dominate fee prices by sending a lot of transactions to each other, and therefore low fees are proposed along the route. To prevent this scenario we want participants to increase B for the calculation of fee proposals of nodes which are not in their social range. We are going to increase B' linearly with every hop greater than the social range, where D denotes the distance from the sender node to the updating edge and SR the social range. Factor k regulates the linear incline.
As a result, proposed fees will be higher for distant nodes, and hence the incentive is given to connect in the social range and not route over long routes.

An example on how to calculate the fee with Procedure 3.4: Under the setup depicted in Figure 3.5 we would arrive at the following two equations $(*)$ for updating edge $e = (3, 4)$. Assuming $B' - \gamma \approx 10$, and $\eta \approx 1$:

$$
\begin{aligned}
T_m(s,4)f((s,3)) + T_m(s,4)f''((3,4))_s &= B' - \gamma \\
7f((1,3)) + 7f''((3,4))_1 = 10 \Leftrightarrow f''((3,4))_1 &= \frac{3}{7} \\
5f((2,3)) + 5f''((3,4))_2 = 10 \Leftrightarrow f''((3,4))_2 &= \frac{5}{5} = 1 \\
f''((3,4)) = \min_{f''(e)}\{f''((3,4))_1, f''((3,4))_2\} &= \frac{3}{7}
\end{aligned}
$$

The choice for the new fee on edge $(3, 4)$ would be:

$$f'((3,4)) = f(e) + (f''(e) - f(e)) \cdot \eta = \frac{3}{7}$$

$$\begin{pmatrix} 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

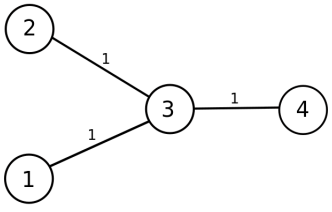

Figure 3.5: From left to right: transaction matrix, and network graph. $T_m(1,4) = 7$, $T_m(2,4) = 5$ and $f((1,3)) = f((2,3)) = f((3,4)) = 1$.

Algorithm 2 implements Procedure 3.4:

Algorithm 2 Fee Update

Input: $G = (V, E)$, T_m, F_m, f
while not all edges e in E are updated **do** ▷ $e = (u, v)$
  $SPs = list()$
  for all s in V **do**
    for all r in V **do**
      if $e \in SP(s, r)$ **then**
        ADD($SP(s, r)$) to SPs
  if $SPs = \emptyset$ **then** ▷ disconnection rule
    $E' = E \setminus \{e\}$, channel parties close channel e
    $F'_m(u, v) = F'_m(v, u) = f'((u, v)) = \infty$
  else
    calculate $f'(e)$ with the help of Proc. 3.4, SPs and T_m
    $F'_m(u, v) = F'_m(v, u) = f'(e)$
  continue with next edge e in E
return $G' = (V, E')$, F'_m, f'

The algorithm is going through all edges of the network graph and proposes a new fee for each, returning a new weight function f' and updated fee matrix F'_m in the end. The order in which edges are being updated depends on the implementation. Individual participants would only consider their own channels being in the edge set. Channel parties should arrive at same fee proposals for common channels.
If an edge has no traffic, i.e., there exists no shortest path using that edge, we want to delete it. In our strategy, any edge without traffic will contribute unnecessarily to the loss (3.2) which we still want to keep at a possible minimum.

After the fee update, we can optimize our network graph again. To update the network, we can use the following Algorithm 3 which is identical to the last part of Algorithm 1.

Algorithm 3 Network Update

Input: $G = (V, E)$, T_m, F_m, f
for all s in V **do**
  for all r in V **do**
    if $c(SP(s, r), f) \cdot T_m(s, r) \geqslant B$ **then** ▷ connection rule
      $E' = E \cup \{(s, r)\}$ ▷ $e = (s, r)$
      s and r decide on channel fee $f'(e)$
      $F'_m(s, r) = F'_m(r, s) = f'(e)$
return $G' = (V, E')$, F'_m, f'

Algorithm 2 and Algorithm 3 can iteratively be applied to a network graph. The ultimate goal is to obtain a network graph in which participants agree on fees being paid on all channels in the network, i.e., our solution converges to a stable network where fee update differences are negligible. To analyze convergence we need a reasonable quantity to express the network state. We use the fee being paid in the whole network as such quantity. We suggest the following scoring function to represent the network state:

$$S(T_m, f) = \sum_{s \in V} \sum_{r \in V} T_m(s, r) \cdot c(SP(s, r), f) \tag{3.5}$$

The scoring function $S : (\mathbb{N}^{n \times n}, f) \to \mathbb{R}$ sums up all fees being paid by all transactions in the network.

Algorithm 4 iterates fee update and network update until a limit of iterations is reached, or differences in the scoring functions are considered negligible:

$$\frac{S^t(T_m, f')}{S^{t-1}(T_m, f)} \leq \epsilon$$

Where t indicates the current iteration step. Note, the weight function is updated in consecutive iterations.

Algorithm 4 Network Convergence

1: Input: $G = (V, E)$, T_m, F_m, f
2: **while** not converged **or** iteration limit reached **do**
3: update fees F_m, f according to Alg. 2 and T_m
4: update network $G = (V, E)$ according to Alg. 3 and T_m
5: **return** $G' = (V, E')$, F'_m, f'

Note, the same transaction matrix T_m is used throughout the iteration otherwise fees could not be decided due to the nature of our fee updating procedure. The convergence behavior is going to be discussed in Section 4.1.

3.2.3 On Entering/Leaving the Network

With algorithms for network creation and network update, we realized a strategy to connect participants in a micropayment channel network and set fees on micropayment channels. We briefly want to mention how new participants can connect to/leave the network. Leaving the network can have different reasons, either a node leaves voluntarily or crashes. For our strategy, the reason does not matter.

Participants entering should connect to the network in the same fashion of Algorithm 1. The first connection should be established to the node where most transactions are going to be sent to. To other receiving nodes the shortest path is calculated, and direct connections created if the costs for sending transactions according to the algorithm are improved. The newly added node can immediately be calculated in for updating fees in the extended network graph containing the node and its newly created edges.
Alternatively, the entering participant tries every possible first connection and eventually finds an optimal solution minimizing his fee payments. This approach needs a longer run time and knowledge about all participants in the network, not only the ones a transaction is going to be sent to, which is somewhat unrealistic.

When a node crashes or leaves, there are several options on how to proceed: either the network is created anew, or a connecting edge is created as the crashed node might have been a connecting node, or in the best case, the network is still connected and can be just updated regularly.
Algorithm 5 proposes a repairing solution if the network should get disconnected:

Algorithm 5 Network Repair (local)

1: network structure changes $\widetilde{G} = (\widetilde{V}, \widetilde{E})$, $\widetilde{T}_m$, $\widetilde{F}_m$, $\widetilde{f}$
2: **while** some nodes $R \subseteq \widetilde{V}$ not reachable **do**
3: connect to unreachable node r in R with $\max(\widetilde{T}_m(s, r))$
4: $E' = \widetilde{E} \cup \{(s, r)\}$ ▷ $e = (s, r)$
5: s and r decide on channel fee $f'(e)$
6: $F'_m(s, r) = F'_m(r, s) = f'(e)$
7: update with Algorithm 4

Participants check with the algorithm if some nodes are not reachable. If so, a connection to the unreachable node where most transactions are going to be sent is established.

Creating the network anew results in a better-optimized network, compared to the other two solutions, but will stall the whole transaction flow by creating and closing channels, which is highly impractical for many reasons.
Repairing the network locally allows the transaction flow which is not influenced by the crashing node to continue. Further, only local timeouts are expected for creating new connecting channels.

For any option, the transaction matrix and fee matrix can be reduced by the sender row and receiver column of the crashed node. For network updates, a crashing node and its channels can be ignored for the time being. If a node returns after a timeout, the network can easily be extended back to the old network state by applying the developed algorithms.

Chapter 4

Analysis

In this chapter, the performance of the developed algorithms for network creation and update is analyzed, especially the convergence behavior of Algorithm 4. All the algorithms have been implemented in the statistical programming language R:[1]

- Algorithm 1, for network creation
- Algorithm 2, for fee update (implements Procedure 3.4)
- Algorithm 3, for network update
- Algorithm 4, for iteratively applying Alg. 2 & Alg. 3

Various parameters are adjustable:

- n: number of participants
- r: number of transactions, e.g., $T_m(i,j) \in [0..10]$
- f_{init}: initial weight function (3.1) on network creation in Alg. 1
- B: blockchain transaction fee cost (on-chain cost)
- γ: how close to B should the fee be determined in Proc. 3.4
- *update_mode*: update edge-wise/node-wise, and in-order/randomly
- ϵ: convergence tolerance in Alg. 4
- i: iteration limit in Alg. 4
- η: step-size in Proc. 3.4

[1] https://www.r-project.org/

In Figure 4.1 an example of an network graph (3.3) created by Algorithm 1 and updated afterwards by Algorithm 4 is displayed. Table 4.1 shows the convergence behavior. The network score is calculated by scoring function S (3.5). Afterwards, all choices of parameters are discussed in order.

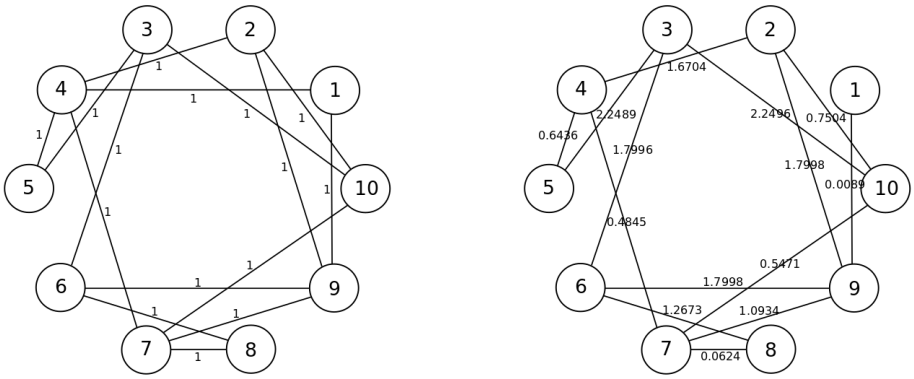

Figure 4.1: Network graph with 10 participants, the initial fee is 1 for all channels and $B = 10$. On the left side is the network graph created by Algorithm 1 depicted. On the right side is the network graph updated by Algorithm 4 with newly calculated fees. Each participant has a unique node ID. Fees are displayed on each edge, and only the first four decimal digits are shown due to readability. Note, edge $e = (4, 1)$ was removed during the update phase.

Iteration	Score	Difference
Initial	198.602933	-
1	181.721089	16.881844
2	177.101706	4.619383
3	178.960943	1.859237
...		
15	181.263752	0.009077
16	181.269899	0.006147
17	181.274053	0.004154

Table 4.1: Convergence behavior of Algorithm 4 applied to the network graph in Figure 4.1. Edges are updated in-order. On average a transaction costs 1.258848.

n = 5/10/20: For micropayment channel networks with 5, 10, or 20 participants, results could be found in a reasonable time frame. These numbers seem insignificant compared to expected real-world networks of thousands or even millions of nodes. Here, we analyzed the whole network on a global scale. Usually, participants will only deal with a fraction of other network participants, and the

algorithms are applied locally by participants.
Nonetheless, the convergence criteria should not change for larger networks, except that the running time is exponential in the number of participants.
We choose the *social range* to be 4. E.g., participant 6 interacts with participants 2-5 and participants 7-10. For shortest paths which route over more than 4 hops to some updating channel, a higher modified on-chain cost B' is therefore used as described in the Section 3.2.2.

r = [0..5]: Each entry in the transaction matrix T_m (3.8) is sampled uniformly random from [0..5]. Parameter r solely scales the amount of fees being paid on each edge but has no further implication on the network structure.

B = 10, f_{init} = 1: The ratio $B/\overline{f_{init}}$ between on-chain cost B and the in average occurring initial fee $\overline{f_{init}}$ decides how high the connectivity in the initial network is. If the ratio is high, the resulting graph will be sparsely connected. Figure 4.2 shows the network creation with $B = 100$ for the same transaction matrix used in Figure 4.1. In contrast, if the ratio is small or even 1-to-1, the resulting graph has an edge between all trading participants.

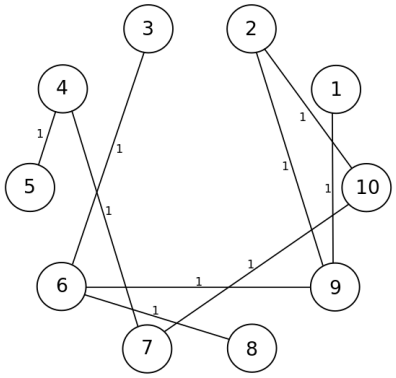

Figure 4.2: Network graph creation by Algorithm 1 with $B = 100$. The network graph has the minimum amount of edges (9) as the initial ratio $B/\overline{f_{init}}$ is higher compared to Figure 4.1.

γ = 0.1: The parameter γ can be seen as a factor which deviates the fee settings by Procedure 3.4, i.e., a certain deviation is added to the already predicted transaction matrix. If more transactions are expected on average, γ can be increased to lower fees in general. Alternatively, γ could also be negative to increase fees as predictions might be overconfident. We use $\gamma = 0.1$ to obtain slightly smaller fees. Varying γ has no impact on the convergence.

***update_mode* = edge-wise in-order**: In all different update modes (edge-wise in-order, edge-wise randomly, node-wise in-order, node-wise randomly) results were converging. Generally, in-order updates converge faster. Random updates could lead to different converging states as the order of updating nodes matters. The convergence behavior is mostly not as smooth compared to in-order updates. In some rare cases, random updates would find a convergence limit faster by chance. We chose edge-wise in-order-updates which yield results in the shortest time frame.

i = 100: We choose $i = 100$ as all found results are converging within this iteration limit.

ϵ = 0.005, η = 0.5: Both parameters ϵ and η influence the convergence of Algorithm 4. We believe $\epsilon = 0.005$ is small enough to conclude convergence. For step-size η we chose the value 0.5. Higher choices will by tendency result in a faster convergence, but fees are set less evenly. With a lower choice of η, a possible limit is determined more accurately but the time for convergence takes longer as update steps are smaller.

4.1 Convergence

All simulations produced by Algorithm 4 were converging. We try to give an explanation. Therefore we first look at one particular fee being updated iteratively with Procedure 3.4. Other fees in the network remain unchanged.

We distinguish between two cases for updating, either a negative (including 0) or a positive minimum is proposed. Note, all fee proposals belong to sender nodes which already have the shortest path routing over the updating edge. Thus, if the fee proposal is negative, the actual fee on the edge will be reduced in any case because the shortest path of the node proposing the negative fee is not going to change. With other fees in the network being static, the fee would converge to 0 during the iterative updates:

$$f'(e) = f(e) - f(e) \cdot \eta, \text{ with } 0 < \eta < 1,$$
$$\lim_{i->\infty} f'(e) = 0$$

Having zero fee is an edge case, which is very unlikely for different reasons. Mainly because nodes, for which a negative fee is proposed, would create at some point a direct connection instead of continuing to send over the unprofitable edge. But nevertheless, the fee converges.
If the fee proposal is positive the minimum is approached:

$$f'(e) = f(e) + (f''(e) - f(e)) \cdot \eta, \text{ with } 0 < \eta < 1,$$
$$\lim_{i->\infty} f'(e) = f''(e)$$

In both cases, additional shortest paths could route over the edge if the new fee proposal is smaller than the original fee, i.e., $f'(e) < f(e)$. Newly calculated in shortest paths will never increase the fee proposal.
We conclude that iteratively updating one edge of a network converges to a stable network state.

Now we want to analyze the convergence behavior for a dynamic network, updated by Algorithm 4, where different update-interleavings are possible, i.e., an update-interleaving describes the change in the proposed fee:

i) from a positive to a negative fee proposal → fee is decreased

ii) from a negative to a positive fee proposal → fee is increased/decreased

iii) from a minimum to a smaller fee proposal → fee is decreased

iv) from a minimum to a larger fee proposal → fee is increased

Again, the shortest path is given for each transaction. The shortest path can change during the update phase if a cheaper alternative route appears. Furthermore, all nodes consider the same transaction and fee matrix (3.9) for updating their fees.

We consider the update of one edge once more and discuss how each interleaving could occur:

i) Due to a fee update, where a positive proposal decreases the channel fee, a new shortest path could route over the edge for which its sender proposes a negative fee.

ii) If the sender who proposes a negative fee finds a better alternative and switches its shortest path, the newly proposed positive fee is either smaller or bigger compared to the actual fee, and increases/decreases the channel respectively.

iii) If due to dependencies a fee somewhere else is increased, i.e., if a fee is increased on some edge, on the updating edge the fee might be decreased if the increased edge fee is used to determine the proposal. In case of an increased fee some precautions, which are discussed in Section 4.2, are necessary.

iv) If the minimum fee proposal of a sender disappears and the new proposal is larger than before. Or again, due to dependencies on other updated edges.

It is worth mentioning that a negative fee proposal could turn into a positive proposal if the fees on a dependent route change accordingly. Mostly, this happens at the beginning of the update rounds.

Because all edges are updated with the same procedure, the difference in updating each fee becomes smaller and smaller. Only when edges are created/deleted, or shortest paths route differently, the successive update round might result in a higher difference, e.g., a newly proposed negative fee on some edge with $\eta = 0.5$ would halve the previously set fee. This behavior could be registered during the performance of the algorithms.

To prevent the network from being unstable (non converging), infinite updating cycles, where the fee is incremented and decremented by some interleavings should not be possible, e.g., interleaving i) and ii) subsequently occur with the same fee proposals. Infinite cycles are not possible because this would mean that some sender would switch to a more expensive route at some point.

Lemma 4.1. *Any network graph, while being updated by Algorithm 4, converges to a stable network state, where fee update differences are negligible.*

Proof. We prove that no infinite cycle of equal fee update differences can exist. We consider a node which switches its shortest path if an alternative path is cheaper regarding fees being paid. For an infinite cycle to occur the fees on an alternative path must therefore always be smaller. Now we know that channels will only set fees almost to 0 in the worst case. With fees being 0 a cheaper path cannot exist and therefore update differences must become smaller and smaller. □

So eventually, the network converges to a stable network state where fee updates are negligibly small. This result is supported by the performance of Algorithm 4.

4.2 Misbehavior

In this section, we loosen our Assumption 3.7 and want to analyze the impact on the network when some participants ignore the proposed strategy and set fees on their channels differently, see Figure 4.3. Ultimately, misbehaving participants want to increase profits.

Other channel parties, which set their fees according to the strategy, will adapt to the misbehaving fee by including it in their calculations if there are any dependencies. Therefore, setting a lower fee will increase, and setting a higher fee will decrease other fees in the network to some extent. If the fee is too high, the network will respond in creating either new channels or routing transactions differently. If the fee is set lower, neighboring nodes will increase their fee, and hence, more transactions are not necessarily attracted. Besides, attracting more transaction for lower fees does not lead to more profit. Nevertheless, there are some setups in which more profit could be made, see Figure 4.4. To prevent

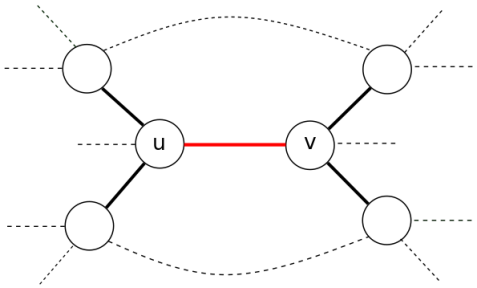

Figure 4.3: Node u and v do not set the fee on their channel, indicated by the red edge, according to the fee updating strategy of Algorithm 2. The strategy is followed by all other participants and also in cooperation with nodes u and v by other participants. Dotted lines represent possible longer indirect routes and connections in the network.

this scenario, participants should check if nodes along shortest paths include all information (transaction count, fees on the shortest path) in their updating procedure. If the fee on some edge is increased, it can be verified whether the fee setting is justified or not. E.g., in Figure 4.4 node i could check if edge (u, v) is considering the 4 transactions of node i and the shortest path from i to j. As the fee is increased, node i knows that the fee is not set according to the strategy and the misbehavior is detected. Node j can perform this check as well.
Since the network structure is driven by the transaction flow, the setup for more sophisticated attacks is even more complicated, i.e., channels must be set up by attackers, and participants will not put funds in a channel they do not want to send over to. Crashing channels with a lot of traffic on purpose can lead to timeouts from which the network can recover eventually with one of our presented solutions in Section 3.2.3.

It can be concluded that ignoring the strategy is not profitable if the network is adapting and participants check if increased fees are justified.

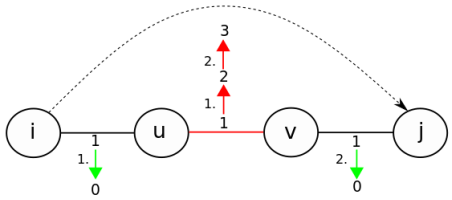

Figure 4.4: Node i sends 4 transactions to node j, with $B = 12$, $\eta \approx 1$. Edge $e = (u, v)$ sets the fee not according to the strategy. In a first step edge $f((u, v))$ is set to 2, as a result $f((i, u))$ is updated to 0. In a second step $f((u, v))$ is set to 3 and $f((v, j))$ updated to 0. In the end nodes u and v earn all fees. Setting the fee $f((u, v))$ directly to 3 would result in node i creating a new edge to node j.

Chapter 5

Results and Discussion

With our algorithms converging to a stable network state, we designed a viable strategy for connecting participants and setting fees in a micropayment channel network.

Since the transaction flow influences the network structure, a centralized network is not favored but also not necessarily prevented. For example, a company which provides products for participants could unintentionally become a centralized hub as a lot of customers want to create a direct connection to make payments on a regular basis. Thus, the strategy does not guarantee a decentralized network.

We want to do a reality check on our model from a technical side. The model provides basic functionalities for a micropayment channel network, i.e., creating channels and sending transactions over the network. To be tried and tested in a real-world network some components are still missing. Firstly, capacities are also important for routing. The capacity of channels could also implicitly determine the initial fee setting and influence the fee update. Secondly, channels in micropayment network are probably directed and have different fees depending on the direction a transaction is sent over the channel.

5.1 Routing

To set our design of the micropayment channel network and the underlying strategy into context, we consider routing transactions in a micropayment channel network. We distinguish between three different routing strategy levels: short-term, mid-term and long-term routing strategy.

The short-term strategy deals with the routing of every single transaction separately, where a decision has to be made within seconds. The sender of the transaction wants to find the best possible route to the receiver node. In our model, this would solely concern the cost of the route being minimal. Participants could also be concerned about the length of the route, where each additional hop is a potential node to fail. The capacity is also a critical factor

for route selection as we have seen. Thus, our strategy also applies least to the short-term strategy.

The mid-term strategy is about determining the fee on a payment channel for which the time range is within minutes. Participants use the developed algorithms to set their fees accordingly. The period in which a fee on a channel will be updated depends on the change in the transaction flow. Fast changing transactions need more frequent fee updates in order to adapt in the best possible way to the network graph (3.3).

In the long-term strategy, participants decide on opening and closing channels. On this level decision and corresponding execution are in the range of hours. Our strategy provides a solution for when to open or close a channel. Differences in the solution found by Algorithm 4 will occur with the change of transaction flow. For different transaction matrices (3.8) and resulting network graphs, a trade-off has to be evaluated. For example, picking the top k channels which are created the most overall different network graphs simulated by our algorithms.

Our strategy applies to the mid-term and long-term strategy.

5.2 One Owner vs. Many Owners

An interesting question for real-world applications arises when considering who actually could implement our micropayment channel design and underlying strategy. We distinguish between the network having one owner who controls the network, and many owners who in collaboration form the network.

For one owner, our strategy practically functions like in our simulation. We have a global view of the users and also a known transaction matrix, for which users indicate how much transactions they want to send to whom. The strategy only concerns the owner. For example, the developers of cryptocurrency X, which implements a fully distributed blockchain, offer the service for entering their micropayment channel network, which provides low fees and immediate payments to other users. The charged fees will pay the developers for maintaining the service. (In this setup users do not need to bother about locked-in funds, which are also managed by the owner.)

As we have seen, the strategy could also work for a majority of participants paying attention to justified fee increases and adapting to decreased fees. But, the border of how many participants follow the strategy and how many not is in a gray area. Thus for many owners, one has to assume that everybody does not follow the strategy and we enter the field of game theory. In this scenario participants, do not only compete with the blockchain, like in our approach, e.g., for fee calculation in Procedure 3.4, but also compete with other channels and routes in the network. Though, obtaining up-to-date information about other

channels and routes might not be that easy (which is an argument to use our design strategy).
The creation of a micropayment channel is still costly. Thus, a minimal connected network is created in a first phase, where participants compete with the blockchain. For every additional created channel a competition emerges. The premise, under which a channel is established, is:

$$\Delta transactions \cdot f(e) \geqslant B$$

s.t. the transaction flow times the fee on the channel is at least greater or equal to the costs for opening the channel. Predicting a future transaction flow on a channel is highly speculative, and profit is not guaranteed (which again speaks for our design). Players in game theory are unaware of the eventual reaction of their opposing players. Therefore, it is predictable that competing channels, once created, will race their fees to the bottom as channels want to steal the traffic from competitors. In the worst case, a fully connected graph with 0 fees on the edges is created.

We end up in a well-known dilemma for which this thesis does not deliver a solution, but, in comparison to the worst case, offers a viable alternative. A crude approximation for an upper bound of the Price of Anarchy (PoA) could look as follows:

$$\frac{\frac{n(n-1)}{2}}{|E|}$$

Where in the worst case a fully connected graph for n participants with 0 fees is created leading to a Nash Equilibrium, and a possible solution from our strategy proposes to create $|E|$ edges leading to a stable network state. For the example in Figure 4.1 the PoA would be $\frac{45}{15} = 3$.
A Social Optimum is achieved by creating only $n - 1$ many channels. Such a network design though is in many aspects inferior to our design as it does not consider anything, e.g., the transaction flow is not considered.

5.3 Related Work

Extensive research has been made to improve micropayment channels in various aspects. Here are some listed providing interesting ideas by which our model and strategy could be extended.

Burchert et al. propose a new layer between the blockchain and the payment channels [9]. By introducing an intermediate layer, they tackle the problem of reallocation of locked-in funds to different channels. In their approach participants need only to be part of a group of shared accounts to perform reallocation. Thereby, transactions on-blockchain are drastically reduced compared to traditionally opening and closing payment channels via the blockchain.

Prihodko et al. came up with a routing algorithm, Flare [3], which could be used for payment routing in the Lightning Network. The goal of the algorithm is to gather for each node information about the network topology. To achieve this, beacon nodes are selected to be able to reach distant nodes in the network. Their algorithm applies to the short-term strategy and complements our strategy.

Rohrer et al. improve route selection by considering payment channel networks as flow networks [7]. Their algorithm provides multiple paths to route transactions of larger volumes.
Their work is also an improvement for the short-term strategy.

Fabrikant et al. introduce a game that models the creation of Internet-like networks by selfish node-agents without central design or coordination [5]. Their approach is based on game theory where the cost for each player in the network is the cost of the total edges laid down by this player, plus the sum of the distances from the node to all others, which is not a criterion in our strategy.

Chapter 6

Conclusion and Further Research

Despite the lack of a real-world network implementation and testing to this end, we managed to develop a strategy which yields promising first results in simulations. The strategy, when applied by micropayment channel network participants, converges to a stable network state in which fee update differences are negligible. Furthermore, fees on channel are chosen in a way such that they are still affordable by participants with a high transaction count, thus, providing an average transaction cost which is much smaller compared to the blockchain transaction fee cost.

We believe that a strategy to set fees on channels is a different approach to improve micropayment channel networks and that our design of a micropayment channel network makes the first step into a new research direction.

Although our model provides basic functionalities, it can surely be extended further. In a next step capacities and undirected edges could be added to the model and the strategy adjusted accordingly.

To simulate our strategy in a network originating from real data, one could analyze the traffic on the blockchain and could try to extract the transaction matrix (3.8), and apply our algorithm to compare resulting costs. This task could be quite challenging because transactions cannot easily be mapped to identities, as arbitrarily many identities can be created.

Bibliography

[1] Joseph Poon, Thaddeus Dryja. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments (2016). [Online; accessed June 2017]. URL: https://lightning.network/lightning-network-paper.pdf.

[2] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System (2008). [Online; accessed June 2017]. URL: https://bitcoin.org/bitcoin.pdf.

[3] Pavel Prihodko, Slava Zhigulin, Mykola Sahno, Aleksei Ostrovskiy, Olaoluwa Osuntokun. Flare: An Approach to Routing in Lightning Network (2016). [Online; accessed September 2017]. URL: http://bitfury.com/content/5-white-papers-research/whitepaper_flare_an_approach_to_routing_in_lightning_network_7_7_2016.pdf.

[4] Chris Pacia. Lightning Network Skepticism (2015). [Online; accessed October 2017]. URL: https://chrispacia.wordpress.com/2015/12/23/lightning-network-skepticism/.

[5] Alex Fabrikant, Ankur Luthra, Elitza Maneva, Christos H. Papadimitriou, Scott Shenker. On a Network Creation Game. *International Symposium on Principles of Distributed Computing (PODC), Boston, Massachusetts, 2003.*

[6] Mike Hearn, Jeremy Spilman. Bitcoin contracts. [Online; accessed June 2017]. URL: https://en.bitcoin.it/wiki/Contract.

[7] Elias Rohrer, Jann-Frederik Lass, Florian Tschorsch. Towards a Concurrent and Distributed Route Selection for Payment Channel Networks. *International Symposium on Research in Computer Security (ESORICS), Oslo, Norway, 2017.*

[8] Christian Decker, Roger Wattenhofer. A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels. *International Symposium on Stablization, Safety, and Security of Distributed Systems (SSS), Edmonton, Canada, 2015.*

[9] Conrad Burchert, Christian Decker, Roger Wattenhofer. Scalable Funding of Bitcoin Micropayment Channel Networks. *International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS), Boston, Massachusetts, 2017.*