University of Applied Sciences and Arts Northwestern Switzerland
School of Business

Effectivenes simulation of a rebalancing algorithm for the Lightning Network under partial participation

Bachelor Thesis

August 1, 2020

Organization	FHNW, School of Business, Basel
Study program	Business Information Technology
Author	Tobias Koller
Supervisor	Prof. Dr. Kaspar Riesen
Project sponsor	Prof. Dr. Thomas Hanne
Expert	René Pickhardt

Abstract

Here follows my abstract Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.1-2

Keywords: Lightning, Bitcoin, path finding

Contents

DECLARATION OF HONOR

I the undersigned declare that all material presented in this paper is my own work or fully and specifically acknowledged wherever adapted from other sources. I understand that if at any time it is shown that I have significantly misrepresented material presented here, any degree or credits awarded to me on the basis of that material may be revoked. I declare that all statements and information contained herein are true, correct and accurate to the best of my knowledge and belief. This paper or part of it have not been published to date. It has thus not been made available to other interested parties or examination boards.

Tobias Koller

Place / Date

Foreword

Some background

1 Introduction

This section gives a brief introduction and overview of the Bitcoin and Lightning technology. It covers the history of digital cash, the advent of Bitcoin and explains the need for an off-chain scaling solution.

1.1 Bitcoin: Peer to Peer Electronic Cash

Bitcoin is a peer-to-peer electronic cash system first introduced in a white paper by the individual or group behind the pseudonym Satoshi Nakamoto (Nakamoto, 2008). This paper lines out the fundamental principles of the Bitcoin block chain that achieves digital transfer of value without a central third party. The next paragraph expands on the pre-bitcoin developments on electronic cash that eventually lead to the rise of Bitcoin. Additionally, we explain why there is a need for additional layer protocols.

glossary
block chain

1.1.1 History of digital cash

Before the digital age cash was the dominant form of payment. A bank note or a coin embodies the respective face value to the bearer of it. Economical transactions can be made by simply exchanging this physical token by which the transaction was immediately settled. However, with the advent of e-commerce this simple and transparent mechanism was no longer possible. New institutions formed to fulfill the need of online transactions. Credit card companies and payment processors filled the gap of trust needed between the sender and the receiver of a transaction over the internet. This architecture came with significant drawbacks. Suddenly, the interacting parties are dependent on third parties which are collecting additional fees. Use of such systems require identification and since the intermediary can track all transactions, this reduces the user's privacy (Narayanan, Bonneau, Felten, Miller, & Goldfeder, 2016).

Bitcoin is by no means the first system introduced to allow for a digital cash system. Already in in 1983 Chaum worked on new cryptographic primitives that should make electronic banking services more private and offer improved auditability (Chaum, 1983). Although his technology still relied on a central "bank" server which issues electronic bills, blinded signatures allowed to anonymously transfer them. In 1989 the company **DigiCash** was found by Chaum to commercialise the idea and a few banks did later implement it. Technological complexity, patents on the invention and the incapability of handling user-to-user transactions prevented it from becoming a success Narayanan et al., 2016.

An important problem in the evolution of digital cash has been the so called **double spend problem**. Digital information has the property of being easily duplicated. This poses a problem to digital cash as this behaviour is generally unwanted. How can a receiver of digital cash be certain that the cash has not been spent to someone else before, thus eliminating its value? Satoshi Nakamoto introduced the concept of a global distributed ledger, a data structure that is append only and where any change must be disseminated to all participants. In order to keep the history of the ledger immutable Satoshi utilizes the idea of a time stamping server first proposed by Haber and Stornetta (1991) in 1991. It works by calculating the hash of a piece of data and publishing it to all the participants. This serves as a proof that the data existed at this given time since otherwise the hash could not have been calculated. The next piece of data to be published also contains the previous hash, effectively linking them and forming a chain. If someone would now want to change the underlying data this would change its hash and since it is included in the next element in the chain also this hash would need to be changed up until the most recent element.

explain hash and add to glossary

Maintaining a global state of transactions in a constantly changing network of participants which can not be trusted is challenging. A single user could spin up hundreds of nodes to overcome the conses of the network. How can a consensus be formed without a central authority? A solution to a similar problem was proposed by Back (2002) in 2002. To prevent e-mail spamming he introduced a mechanism that requires the sender to solve a puzzle that is computational heavy. This so called "proof of work" is requested by the receiver of the e-mail to trust that it is no spam. Since this computation can easily be done for one e-mail it becomes a big burden to do for thousand of e-mails therefore avoiding spammers. The puzzle simply involves finding a value whose hash starts with a certain amount of zero bits. Since the result of a hash function can not be predicted only brute force can be applied to find such a value. By selecting the number of leading zero bits one can change the difficulty of the puzzle. Each additional zero leads to the difficulty to be doubled. In order to add transactions to the Bitcoin ledger a participant constructs a block consisting of transactions, computes the proof of work and publishes it to the network. Only if all transactions are valid and the work done has been verified network participants append the transactions to their local copy of the ledger and further broadcast the block.

add pow to glossary

Combining proof of work with the chaining of hashes introduced in the last paragraph results in a strong security model. An attacker who wants to change the history of the ledger would need to recompute the proof of work of the changed and all subsequent blocks as they are linked by their hashes. Therefore, every new block makes it increasingly more difficult to change a transaction in the ledger. Number of blocks on top of a transaction in question is hence be referred to as **confirmations**.

1.1.2 Scaling solutions

One of Bitcoin's value propositions is being decentralised. At the same time every transaction ever made must be distributed and stored among all network peers. It becomes obvious that some trade-off has to be made to maintain those two properties: scalability. This is often referred to as the scalability trilemma which states that in distributed systems the three objectives **security**, **decentralisation** and **scalability** can not be achieved in full extent at the same time. While two can often be achieved, there are certain trade-offs to be made in the third domain. This section is explaining why this is true for Bitcoin and what main categories of solution exist.

trillema in glossary

While decentralisation can be described on many levels we focus here on the decentralisation of validating nodes. Those are network participants that verify the blocks published by miners. Decentralisation would be best achieved if every user of Bitcoin would run its own fully validating node to receive information about the ledger independently. On the other hand the network would be centralised when only few nodes would validate and users would need to trust those to tell the truth about the ledger state. To keep decentralisation high it is crucial to keep the hardware and network requirements as low as possible. The Bitcoin protocol restricts the amount of data to be processed to one block of 1 megabyte per 10 minutes. A full node in the network must be able to download at least 1 MB / 10 minutes in order to keep up with the tip of the chain. Lower bandwidth would cause it to get left behind without every being able to catch up. Additionally, the full node must keep the full ledger on storage. This yields to approximately 286 GB „blocks-size", n.d. at time of writing, increasing linearly in the future. This upper limit block size results in a throughput of approximately 7 transactions per second

add full node to glossary

(tps) Poon and Dryja, 2016. Clearly by several orders of magnitude smaller than what it would require to become a world wide payments network.

Bitcoin, by design, promotes security and decentralisation while sacrificing scalability. However, to be usable by everyone the scalability issue needs to be addressed. As Bitcoin is not owned by anyone there is no one party to decide on the future design decisions. This lead to a scaling debate with two ideological camps on how to progress. Scaling on-chain or scaling off-chain.

Scaling on-chain means lifting the 1 megabyte block limit, allowing for a higher transaction throughput. While this seems the most straight forward solution it can not be achieved without trade-off. As previously discussed decentralisation can only be maintained by keeping the hardware and network requirements low. Removing this restriction to allow worldwide usage would mean that nodes soon need to process hundreds of megabytes or even gigabytes per second, effectively reducing the number of nodes that can still keep up, leading to a more centralised network.

An off-chain scaling solution describes any system that acts outside of the Bitcoin protocol but is linked to it, in a way that leverages the number of economical transactions that can be performed per single on-chain transaction. These solutions build a second layer of abstraction. While still using functionality of the base layer they can reduce their dependency and make their own design decisions and trade-offs based on the scalability trilemma. The Lightning Network is only one possible off-chain solution and is described in the next section in more depth.

glossary second layer

glossary base layer

1.2 Lightning technology

The Lightning Network is a network protocol that utilizes Bitcoin as its underlying trust system. It can, therefore, be described as a “second layer” protocol building upon the Bitcoin “base layer”. Bitcoin does not scale on the base layer since it was designed with security and decentralisation in mind. Off-chain solutions like Lightning are developed to facilitate more transactions on a different layer without compromising the properties of the base layer (Poon & Dryja, 2016).

1.2.1 Payment channels

The transaction bottleneck in Bitcoin is imposed since every network participant needs to be updated about every transaction. This is required to ensure the integrity of the system and not because the transactions are of interest to the nodes. In fact they can learn very little about the parties involved in a transaction as pseudonymous public keys are used as identifiers. Poon and Dryja mention in the Lightning Network Paper that as long as only two participants care about a recurring transaction there is no need to inform the entire network about it (Poon & Dryja, 2016). They therefore propose that those two participants do not sent the transaction to the network but instead hold on to it and agree on their balances bilaterally. „Micropayment channels create a relationship between two parties to perpetually update balances, deferring what is broadcast to the block chain in a single transaction netting out the total balance between those two parties.“ (Poon and Dryja, 2016, p. 4)

Opening such a payment channel requires the two parties to create an on-chain Bitcoin transaction which spends an amount of bitcoin to a 2-of-2 multisignature contract. Only both parties collaboratively can spend it from there. At the same time they create a refund transaction that spends from the multisignature contract and distributes the current balance state. However, they could broadcast this transaction instead they can also decide to hold on to it and update it at a later stage. Holding the signed Bitcoin transaction ensures that they could at any time get their balance back on-chain. Every time they wish to transact with each other they just sign a new refund transaction which distributes the updated channel balance.

Example: Assume Alice and Bob open a payment channel and both send 0.5 BTC to a 2-of-2 multisignature contract. This channel has a total capacity of $0.5 + 0.5 = 1$ BTC. At the same time they also create a transaction that pays them back their initial balance of 0.5 BTC each. This transaction is not sent to the block chain but kept private by the two parties. If Alice wants to send Bob 0.2 BTC they update this transaction spending the initial 1 BTC to be distributed as follows: 0.3 BTC to Alice; 0.7 BTC to Bob. This payment was only agreed upon Alice and Bob, no one else needed to be notified. Unlimited future transactions like this can take place as long as the sender still has some balance. At any point in time Alice or Bob can decide to broadcast the latest transaction, distributing the agreed balance and effectively closing the payment channel. As depicted in figure 1.1 there can be many modifications to that balance.

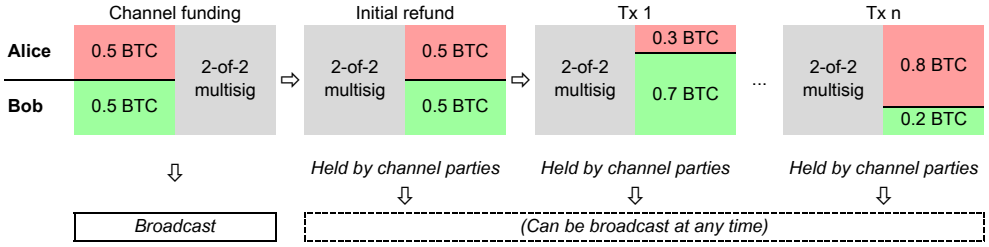


Figure 1.1: Life cycle of payment channel.

Whenever the parties decide to end there collaboration they can close the channel by broadcasting a closing transaction to the block chain which pays out each party its respective balance. While beneficial, collaborative channel closure is not required, each party can close the channel at any given time by broadcasting the latest transaction agreed upon.

Participants in such channels are referred to as Lightning nodes, a computer system that runs an implementation of the Lightning Network protocol. Those nodes are not to be confused with Bitcoin nodes. However, since most Lightning nodes need direct access to the Bitcoin block chain most Lightning nodes are Bitcoin nodes at the same time.

1.2.2 Using other channels for payments

Since a payment channel is a biparty relationship an update of channel balance can only ever represent a transaction between those parties. Instead of opening a channel to each participant a node wants to transact with there is the opportunity to route payments through multiple channels.

Often economical transactions to a specific recipient are made only once. It would therefore, defeat the purpose of Lightning to open a channel for this unique payment because both opening and closing of a channel takes each one transaction on the base block chain. Using Lightning would not use the load on-chain but rather increase it by a factor of two. This demonstrates that a payment channel should only be opened if the cost of doing so can be amortised over many transactions. This is either the case if the channel parties are expected to repeatedly transact with each other or if they can facilitate transactions between other nodes in the network through routing.

glossary
econ. tx

Handling payments off-chain so far only worked between two parties that update Bitcoin transactions off-chain with the security of claiming their balance at any time on-chain. How can

payments through multiple channels be accomplished without introducing trust into the system? If Alice wants to pay Carol through Bob, how can she be ensured that Bob will not keep the money and refuse to pay Carol? The solution are so called **Hash Time Locked Contracts** or short "HTLC" .

define htlc in glossary

First Alice asks the payee, Carol, to create and keep a secret **R** and only share the hash of it, $hash(R)$. Alice then uses this, so called *preimage* , to create a special HTLC transaction that promises Bob to receive the payment amount if he has knowledge of R. Bob also gets informed by Alice who is next in the path to find out this secret. Bob then himself creates an HTLC with the same preimage to Carol offering her the same amount upon disclosure of R. Carol is the payee in the transaction and has knowledge of R so she could technically claim the funds on-chain. There is however an easier solution than this. After she releases R, Carol and Bob can simply agree to update their channel to reflect this payment. The same is done between Alice and Bob leading to the state where all channel balances are updated and the HTLCs is rendered useless. This mechanism is called "time locked" because the offer of payment to the knowing R is only valid for a certain timespan. If R does not get released until the defined expiry the HTLC turns invalid. Therefore, it is important that each successive HTLC has a lower timespan.

glossary preimage

1.2.3 Routing

The previous chapter explained how a payment can utilize multiple channels to reach its destination without the introduction of trust between participants. This section describes how routing takes place and what trade-off needs to be made.

To ensure privacy of payments the routing protocol uses onion routing . The sender of the payment encrypts the information in multiple layers so that each hop only learns about the next hop in the path. Each intermediary node only knows where the payment comes from and where it goes next. Who is the payer and the payee remains secret. Only the last node in the path finds out that it is the actual receiver of a payment.

define onion routing in glossary

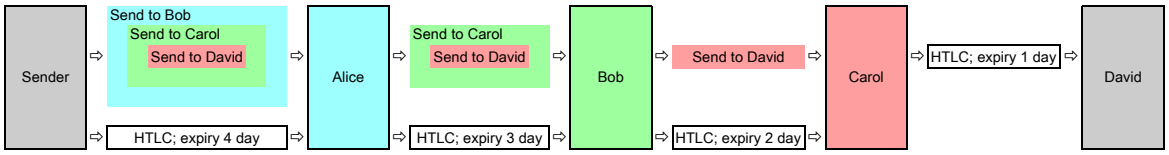


Figure 1.2: Onion routing preserves privacy over the chose path.

Figure 1.2 color codes the information which is only readable by the respective node with the same color. In each step the outermost layer is decrypted and the next hop gets revealed. The remaining encrypted data is passed on down the line. This ensures that each node only has a local view of the path knowing only about its predecessor and successor. Thus follows, the source of the payment needs to determine the complete path before sending a payment.

Sending a payment requires the sender to find a viable path to the recipient. This is called source routing and necessitates some information about the channel graph. When a new channel opens the parties notify the network about the channel, its capacity and the fee policies. Each other network participants stores this information and builds up its local view of the network. When a node constructs a payment it can query a path with enough capacity along all its channels, though this is not enough to be certain that the payment can be routed. The payment in

glossary source routing

figure 1.3 of 0.5 BTC can not be routed from Bob to Carol as Bob only owns 0.2 BTC of the 1 BTC channel between him and Carol. This information however is only known to Bob and Carol, not to Alice. Alice only sees the public information which tells her that the channel has 1 BTC capacity. Local channel balances are kept private for two reasons. Every payment over a channel makes its balances to move. Keeping all nodes updated about all channel balances would necessitate to update them about every transaction. This is exactly how Bitcoin on the base layer work and the reason it does not scale. Secondly, informing the Network about every transaction is very bad for privacy, as an observer could easily determine who pays to whom over what path.

Not knowing local balance distribution of channels makes it hard for source nodes to construct a reliable path. When a payment fails along the path a new route can be calculated and tried. This however is coupled with time delays that can cause a negative user experience.

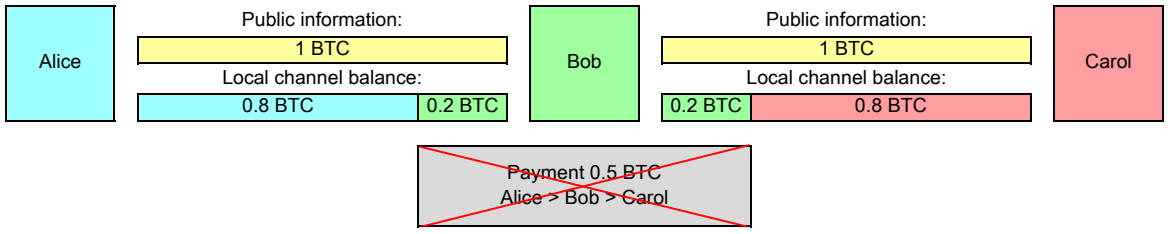


Figure 1.3: Payment fails when channel balances cannot accommodate the required transfer.

2 Problems in graph theory

Graph theory is an old discipline which can be used to solve multifaceted problems in the real world. The following chapter contains an introduction to the basic principles and terminologies. It then defines different problems for which graph theory can be used to find solutions. More emphasis is given to the topics which are applicable to the Lightning Network.

2.1 Formal definition

The following paragraph defines the fundamentals of graph theory as described by Rosen (2012). A graph $G = (V, E)$ consists of a set of vertices V and edges E. An edge consists of two vertices which are called its endpoints. In an undirected graph the endpoints are unordered, whereas the pair of vertices in a directed graph are ordered. The edge (u, v) of a directed graph starts at u and ends in v. In a **simple directed** graph each directed edge can appear only once. When multiple edges go from one start vertex to the same end vertex the graph is named **directed multigraph**. An edge that connects a vertex to itself is called a loop. Assigning weights to edges allows for more complex representations mere connection and is useful if the real world representation of a connection has certain **cost** or **distance** that one might want to minimise (maximise).

Go from general graph theory problems into more specific areas, 1 paragraph per topic

Explain why finding a path in the lightning network can be so difficult

2.2 Overview of broad problems

write an introduction to this section

In **graph coloring problems** vertices can be thought of areas on a map. Areas that share a border are connected with an edge. The goal is to use as little colors as possible to color each area / vertex in such a way that adjacent areas have different colors. This minimum number is called the *chromatic number* $\chi(G)$.

Certain properties of graphs can only be proven if they are fulfilled in all their **subgraphs**. Hence, it is important to find those. A graph $H = (W, F)$ is a subgraph of $G = (V, E)$ when $W \subseteq V$ and $F \subseteq E$. A subgraph can be **induced** by $W \subseteq V$ which means the edge set F contains only edges of E for which both endpoints are contained in W. Similarly the union can be built of two graphs G_1 and G_2 where $G = G_1 \cup G_2$ and $V = V_1 \cup V_2$.

Another area of graph theory aims to determine whether two graphs are **isomorphic**. This means „there is a one-to-one correspondence between their vertex sets that preserves edges.“ (Rosen, 2012, p. 668) Different algorithms are being developed that can determine isomorphism of two graphs, however with exponential worst-case time complexity. These algorithms mainly find application in chemistry and bioinformatics.

Different **route problems** exist where the solution tries to find a path with certain properties within the graph. In the “seven bridges of Königsberg” the four sections of land are separated by a river and seven bridges are connecting them. The problem of passing every bridge exactly once and returning to the start can be formulated in terms of a graph with sections as vertices and bridges as edges. It is the question whether a graph has an Euler circuit or not. Many real-world applications exist where each edge needs to be visited once.

When edges have a weight assigned different problems can be modelled and optimised by finding a shortest path. A famous algorithm to find such a path is “Dijkstra’s Algorithm”. Similarly, in the Traveling Salesperson Problem each vertex needs to be visited exactly one before returning to the starting point. Algorithms finding a path with low cost often use approximation since no algorithm with polynomial worst-case complexity exists.

Different problems deal with the **flow** in a network. Such networks have a capacity assigned to each directed edge. Often the goal is to find the maximum flow between a source and a sink node such that all intermediary nodes have equal in and outflow (Even & Tarjan, 1975).

2.3 Application in the Lightning Network

The Lightning network can best be described by a **simple directed graph**. That is a graph with directed edges without loops and multiple directed edges. Since a payment channel between u and v allows payments to flow in both direction it can be modelled by use of two edges (u, v) and (v, u). In terms of network flow theory the capacity is the amount that can flow from u to v which in Lightning is defined as the local channel balance of u. Figure 2.1 visualises a channel between Alice and Bob with 1 BTC capacity of which Alice holds 0.2 BTC and Bob holds 0.8 BTC.

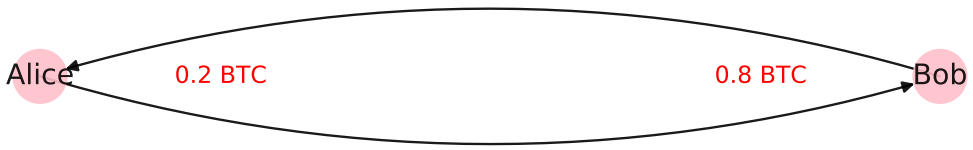


Figure 2.1: Lightning channel represented as simple directed graph..

2.3.1 Fees

So far the properties of a network graph could be mapped to a Lightning payment channel. There is another concept within lightning that need to be accounted for. When payment channels get used by other nodes than the two channel partners they can ask for fee to be paid. This can be compared to a bridge toll that needs to be paid by everyone crossing the bridge. It is a compensation for the capital that the channel owner has "locked". The fee consists of two parts, a fixed fee and a variable fee. While the fixed fee is the same for every payment, the variable increases linearly with the amount being transferred.

2.3.2 Path finding problem

As laid out in section 1.2.3 the initiator of a payment needs to construct the path before sending the payment. Nodes trying to find such a path work with limited information. While they know what channels are available and what their capacities are, they do not know about the balances and therefore whether the nodes can forward their payment or not. Hence, it is likely that a payment attempt fails because a node had insufficient balance. The paying node needs to find another route and retry the payment until it succeeds. If the payment fails repeatedly it can cause delays that are bad for the user experience.

This problem gets amplified because in the current protocol channels are single funded. This means one party provides 100% of the channel capacity. A collaborative approach to funding a channel might be possible in the future. As a consequence the funder of the channel owns initially 100% of the channel balance and the other side 0% which means the channel can only be used to route into **one direction**. A third party trying to route through this channel has a 50:50 chance that he or she can not even route a minimal amount of 1 satoshi .

define satoshi and other units of btc

The Lightning path finding problem can best described by a **flow network** . The goal of a node constructing a payment is to find a path that has enough capacity along its channels to route to the destination. Rules from flow networks apply according to which each node, except source and sink node, must have equal amounts flowing in as are flowing out.

glossary flow network

In the next sections terminology native to the Lightning protocol is used. I refer to the directed graph as the **network**. Vertices are called **nodes** and a pair of opposite directed edges are a **channel**. The **capacity** defines the **total amount** of a channel which is distributed between the two node's balance, as opposed to the amount which can flow in flow networks. This flow is determined by the node's **balance**. The terms **route** or **path** are used interchangeably and are defined by a set of channels that can be traversed. **Fees** is the general term that includes both **fixed fees** and **variable fees**. Fee policies are set by each node individually for every channel. This means a channel can charge different fees in either way as both nodes set the fee policy for their balance. **Rebalancing** is the activity of routing a payment in a circle back to the initiating node and is used to reallocate balances within the set of channels a node controls.

2.4 Previous work

Pickhardt's and Nowostawski's publication "Imbalance measure and proactive channel rebalancing algorithm for the Lightning Network" (Pickhardt & Nowostawski, 2019) serves as a base to formulate the question for this thesis. In their work, they present a solution for the pathfinding problem in a privacy-aware payment channel network. The proposed solution includes a rebalancing protocol which the nodes of the network should follow to achieve a higher balancedness (for itself but also the entire network). It consists of instructions to proactively rebalance their channels within their friend of a friend's network, redistributing the relative funds owned in a channel but leaving total funds owned unchanged.

Rebalancing is an activity where one node engages in a circular payment that pays itself. This is only possible when the node has at least two channels with different peers. The payment gets routed **out** through one channel and is **received back** over another. On the way, it can use one or more hops to find back to the sender node. This procedure enables a node to change the balances of the individual channels while the total node balance stays the same. In practice, there would be a fee collected by the intermediate nodes whose channels are used. In the proposed rebalancing protocol nodes would forego the fee and only participate in the rebalancing attempt if their balancedness improves as well. The rebalancing algorithm is described in more detail in the section .

include ref to experiment

2.5 Problem statement

These payment channel networks are decentralized by nature and protocol changes can not be forced upon the node operators. Therefore, the question arises on how effective this protocol change will be assuming only partial participation of nodes. What are the effects of different levels of participation on the imbalance measure [1] of the network during repeated rebalancing cycles? What is the effect of different levels of participation on the network's ability to route payments between random nodes?

The next part of the paper defines the rebalancing algorithm proposed in a protocol change. Then an experiment is constructed which mimics the real Lightning Network with all its participants and payment channels. I formally define multiple performance measurements which are later used to measure the improvement of the network's ability to route payments. Different

[1] Defined as the inequality of the distribution of a nodes channel balance coefficients

scenarios are then simulated under different levels of participation also testing different ways to select the participating nodes.

3 Proactive Rebalancing Protocol

Pickhardt and Nowostawski proposed in their paper a rebalancing algorithm which when proactively executed would improve the overall networks ability to route payments (Pickhardt & Nowostawski, 2019). The next sections define the algorithm in more detail and formally defines the protocol change. In later section 4.4 we show how it is implemented as part of the simulation.

3.1 Rebalancing

Figure 3.1 shows an example network with four nodes and four channels that are recently opened. The capacities are hence all on one side of the channel. Such an allocation of balances limits the use of the channels severely as they can only send payments into one direction. If Alice wants to have a better balance among her channels she can start a process called **rebalancing**. This is a self-payment which routes an amount from one of her channels through the Lightning network to another of her channels. In this example Alice decides she has excess balance in the channel with Bob and insufficient balance in the one with David.

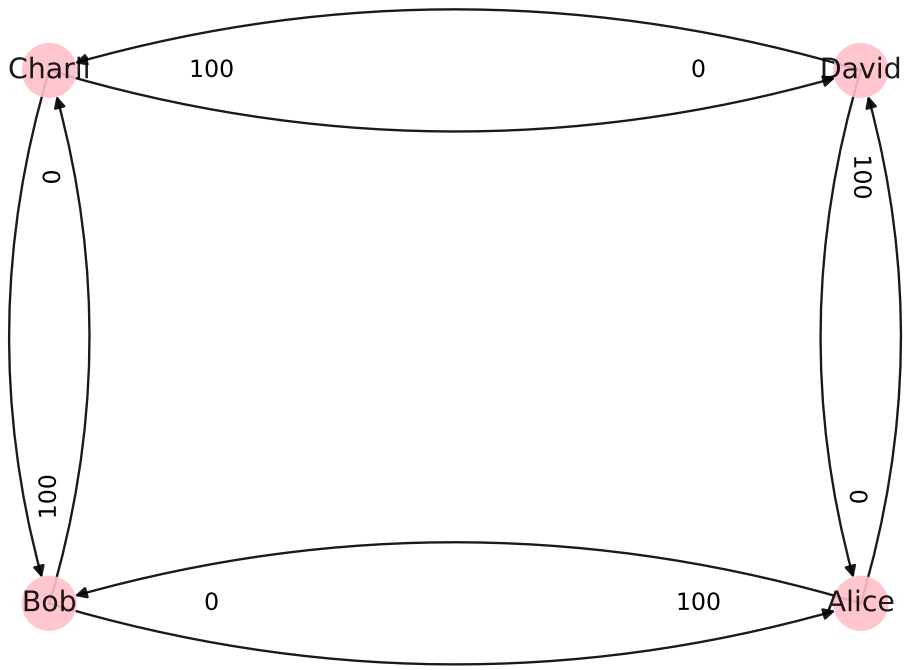


Figure 3.1: Initial state of newly opened channels.

She can now construct a payment of e.g. 40 with the path [Alice, Bob, Charli, David, Alice]. This reduces her balance in the channel with Bob by 40 (new 60) and increases her balance in the channel with David by 40 (new 40). Both of her channels are now better balanced and allow for more payments to be routed. The resulting state is shown in figure 3.2 and the path of the rebalancing payment is indicated in red.

This action performed by Alice did of course not only change the balances of the channels she is involved but also the ones she used to route the self-payment. In this case the channels between Bob-Charli as well as Charli-David are also more balanced. While this was the case in this

example it is not guaranteed to be true in all cases, it might well be that some channels end up less balanced. The reason why all the nodes are incentivised to participate are fees that they can collect on each payment they forward. In fact they do not even know that this payment is a rebalancing transaction. Hence, Alice has to pay fees for this rebalancing operation.

Rebalancing can be summarised as an activity performed by a single node attempting to improve the local balances of its channels by sending a payment out through one channel with excess balance and receiving it through a channel with too little balance, effectively paying itself. For all the nodes in the routing path this payment looks no different to any other payment, so they charge their normal routing fees. While the balances within the channel change a node's total balance remains constant. Alice still has a total balance of 100 allocated differently among her two channels.

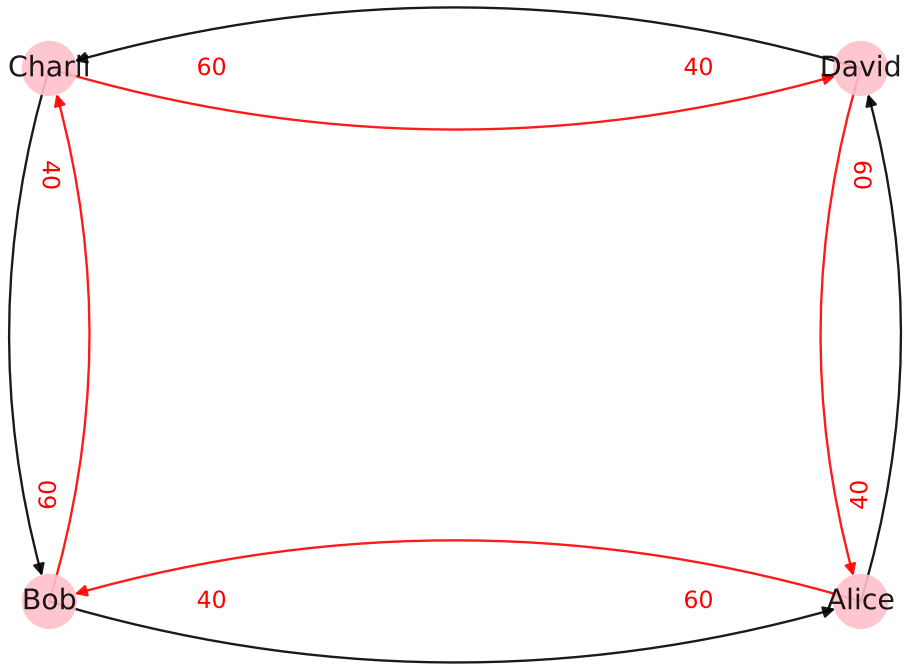


Figure 3.2: Rebalancing from Alice through Bob, Charli and David.

3.2 Rebalancing algorithm

The previous section was about how nodes can *improve* their channel balances. While the improvement in the example was quite clear it has to be stated what the ultimate goal is towards which the nodes try to improve. Intuitively, a distribution of 50:50 in each channel could be thought of as an appropriate goal. In reality this is not a feasible target.

3.2.1 Imbalance definition

The following terms will first be defined to better understand the problem. The two nodes in a payment channel $e = (u, v)$ are denoted as e_u and e_v. The capacity of $e = (u, v)$ is privately split among the two nodes channel balance $b(e_u)$ and $b(e_v)$. Therefore, the capacity of the channel $c(e) = b(e_u) + b(e_v)$. The **channel balance coefficient** for u represents the ratio

a node u has in a channel $e = (u, v)$ and is defined as $\zeta(u, v) = \frac{b(e_u)}{c(e)}$. Respectively, for the node v the coefficient is $\zeta(v, u) = \frac{b(e_v)}{c(e)}$. As both balances make up the total capacity it follows $\zeta(u, v) + \zeta(v, u) = 1$. Let us define all the channels node u is part of as $U = n(u)$. The **total funds** the node u owns can be denoted by $\tau_u := \sum_{e \in U} b(e_u)$. The **total capacity** of the node's channel $\kappa_u := \sum_{e \in U} c(e)$. We can now define the **node balance coefficient** as $\nu_u = \frac{\tau_u}{\kappa_u}$.

Both τ_u and κ_u must remain constant during rebalancing operations. This is because the rebalancing payment pays always the node itself, so the **total funds** of a node can not increase or decrease. As no new channels are opened or closed and capacities in existing channel can not be changed **total capacity** must remain constant. The initially stated goal to have a 50:50 distribution in all channels is only achievable with a **node balance coefficient** of 0.5. As this coefficient remains constant only nodes that already have a node balance coefficient of 0.5 could reach that state.

Pickhardt and Nowostawski had to come up with a different balance measure which is applicable to all nodes no matter what node balance coefficient they had. So they generalised the rule discussed in the previous chapter. Every channel of node u should have the same **balance coefficient** $\zeta(u, v)$ which would then equal the **node balance coefficient** ν_u. In other words, making all channel balance coefficients as similar as possible. They then defined a measure to assess how well this is achieved for a particular node. The Gini coefficient measures the inequality of a distribution. For a node u with channel balance coefficients $\zeta_{(u,v_1)}, \ldots, \zeta_{(u,v_d)}$ they define $G_u = \frac{\sum_{i \in U} \sum_{j \in U} |\zeta_i - \zeta_j|}{2 \sum_{i \in U} \sum_{j \in U} \zeta_j}$. When all balance coefficients are equal the Gini coefficient G_u turns to 0. A value of $G_u = 1$ represents the most unequal distribution (Pickhardt & Nowostawski, 2019).

3.2.2 Proposed algorithm

The proposed algorithm should be executed by each node individually. The main goal of each node is to achieve a low value of imbalance G_u. The node can improve its imbalance by repeatedly executing the following steps according to Pickhardt and Nowostawski, 2019:

1. A node u computes its node balance coefficient ν_u.
2. u then computes its channel balance coefficients $\zeta_{(u,v_1)}, \ldots, \zeta_{(u,v_d)}$.
3. All channels $e = (u, v_i)$ for which the channel balance coefficient is higher than its node balance coefficient, are selected as $C = \{(u, v_i) | \zeta_{(u,v_i)} - \nu_u \ > 0\}$.
4. From the candidate set C a random channel $e = (u, v)$ is selected.
5. Now the node searches for a circular payment to itself along $e = (u, v)$ by choosing a path $p = [v, x_1, \ldots, x_n, u]$. The amount of that payment should decrease the value of $\zeta_{(u,v)}$ to that of ν_u and can be computed as $a = c(e) \cdot (\zeta_{(u,v)} - \nu_u)$. The end of the circle should be a channel (x_n, u) for which the channel balance coefficient $\zeta_{(u,x_n)}$ is smaller than the node balance coefficient ν_u.
6. The node conducts the payment if all the nodes on the path p agree to participate. It could happen that some nodes will only participate with a value smaller than a. As this is already progress u will accept the suggested amount instead of being stubborn.
7. Repeat all steps as long as the local balance coefficients are not even enough and as long as paths are found.

3.3 Protocol change

Rebalancing itself is already supported in the current Lightning protocol as it includes normal payments where the sender and receiver is the same entity. However, the incentive to do so is very small. The initiating node is the one who pays the fee for the payment. A node would only do such transactions if it can expect to earn fees on the rebalanced channels that exceed the amount of fees payed for the rebalancing.

Pickhardt and Nowostawski propose a change in the protocol to make rebalancing more explicit (Pickhardt & Nowostawski, 2019). As the proposed algorithm in the previous section 3.2.2 shows, only nodes would participate that also benefit from the rebalancing (step 6). As this would be a win-win-situation for all participating nodes they could agree to forego the fees. This change towards a **fee-free rebalancing** would incentivise nodes to do this proactively as soon as they see the opportunity to improve their balance.

Additionally, the recommend to adopt a change in the communication protocol which allows „the network to work collaboratively towards achieving a good balance by locally sharing rebalancing hints“ (Pickhardt and Nowostawski, 2019, p. 8). This sharing of local channel balances would only need to be done among the local neighbourhood of a node. This is where the nodes should look for rebalancing circles and providing hints advertises willingness to rebalance in a certain direction without revealing the exact balance of the channel in question. The privacy drawbacks are minimal as balances in the local neighbourhood can easily be probed anyways (Tikhomirov, Pickhardt, Biryukov, & Nowostawski, 2020).

3.3.1 Reasons to not participate

So far the presented algorithm sounded like a win-win-situation for everyone and there would be no discussion around adopting it. However, it is not that simple. There are different reasons why nodes would not want to participate in such a change. Some nodes might not want to forego the revenues in fee. If they decide not to participate but most other do, they still benefit from the better overall balance of the network. So, without contribution they can harvest the benefits of others who participate.

Should I list the need of nodes to not follow the proposed Gini-approach? Because they could technically follow a different startegy and still do fee-free-balancing and hint-sharing

4 Experiment

4.1 Methodology

This section gives an overview over the technology used and the classes created to run the experiments described in later sections. According to Al-Taie and Kadry Python is a general-purpose tool that becomes more and more popular in data science and graph analysis. There are many open source libraries available in the domain of analysis of complex networks, statistics and the visualisation of data (Al-Taie & Kadry, 2017). Furthermore the experiments in the previous paper were conducted with Python.

4.1.1 Python libraries

NetworkX is a Python library that offers tools for to create, manipulate and study the structure, dynamics and function of complex networks (Al-Taie & Kadry, 2017). The library provides basic

operations with different types of network and also implements common algorithms often used in the study of graphs.

For all kinds of visualisation the package Matplotlib is used. It offers broad functionality to create various types of charts. Also NetworkX has some basic drawing capabilities to visualise graphs and relies on the Matplotlib package. In this thesis visualisation of graphs is not of great use and Matplotlib is mainly used for the generation of data plots (Al-Taie & Kadry, 2017).

Numerical computation and statistical calculations are performed using the software packages NumPy and SciPy.

In order to reproduce the results presented in this work all Python dependencies are written down in the file `04_Simulation/requirements.txt` on the Github repository [2] and the `README.md` file demonstrates how to set up the execution environment.

4.1.2 Network class

While NetworkX provides the basic graph functionality, I need more logic to build all the Lightning specific functions. The **Network** Python class holds all these features of which I give rough overview here. All mentions of **Network** in this section refer to the Python class named like this.

Most importantly the Network is a container that holds two NetworkX graphs. The constructor of the Network class in listing 1 shows these two variables `G` and `flow` which will be explained in more detail later.

```
1 class Network:
2     def __init__(self, G, cycles4=None):
3         self.G = G
4         self.flow = None
5     ...
```

Listing 1: Part of Network's init method.

Figure 4.1 visualises the states a Network can be in and how a Network can be exported and restored from file. The first state is a Network that went through all the preprocessing and is ready to start with an experiment of any kind. It serves as starting point for all further modifications. To ensure reproducibility of the results it is important that the same initial state is used all the time. For the very first time, this state can only be reached by parsing a channel file (`parse_clightning()`) and going through the preprocessing. The content of such a channel file and all the preprocessing steps are explained in sections 4.2 and 4.3. After this state is reached one can create a snapshot (`create_snapshot()`) which is stored to a file. In addition a fingerprint is generated and used as identifier for this specific network state. The fingerprint is calculated by hashing all the networks nodes, edges and attributes and prevents accidental load of any altered state.

Once a snapshot is generated the Network object can be recreated from there without going through the preprocessing again. The `restore_snapshot()` function takes the fingerprint as parameter and looks for the corresponding file to reload the state.

[2] https://github.com/TKone7/ln-bachelor-thesis

From the initial state one experiment can be executed. Such an experiment changes the underlying network graph and stores results within the Network object. The results should then also be exported with the `store_experiment_result()` function. It allows to later restore this specific experiment by calling the `restore_results_by_name()` class method. This setup allows to run an time intensive experiment once and then create many evaluations of the result without recomputing the experiment.

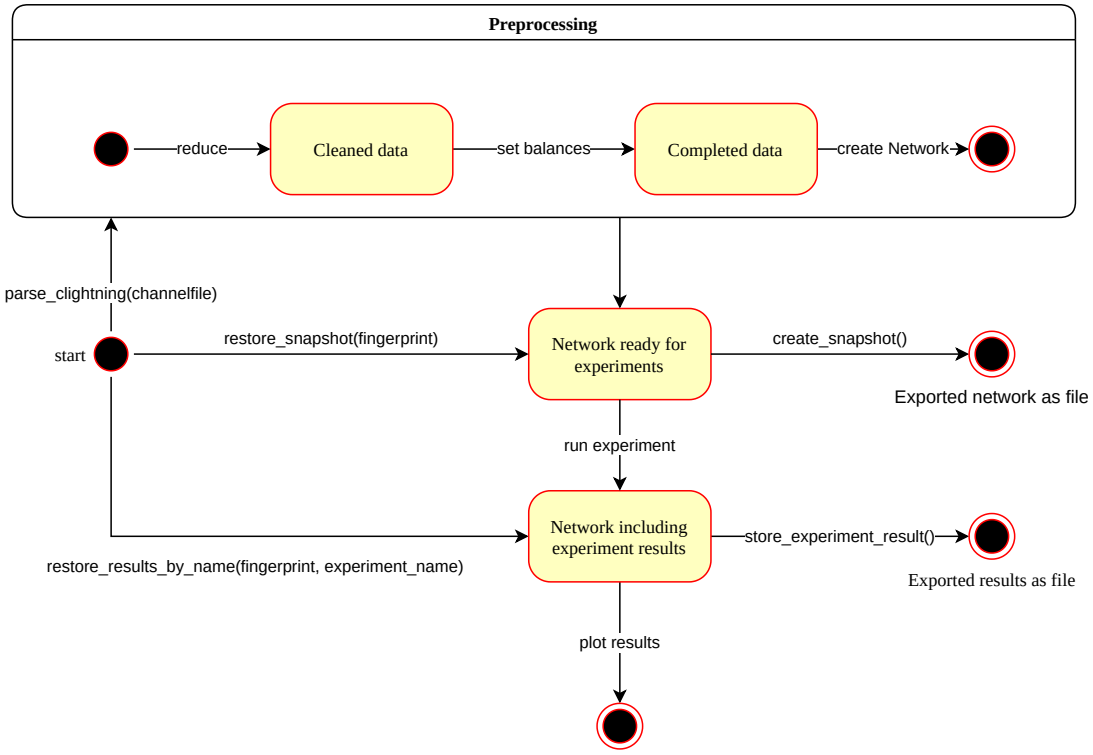


Figure 4.1: State diagram of the Network object.

4.1.3 Experiment class

Many of the experiments actually need several Networks to perform experiments with different parameters. The experiment class is used to interact with those Network objects. This class contains also the definition of an experiment type and takes the parameters as arguments. The different *setup*-methods define the types of experiments available. After the experiment is setup the generic `run_experiment()` and `plot_experiment()` can be called. Figure 4.2 illustrates the relationship between the classes and shows the public methods of the Experiment class.

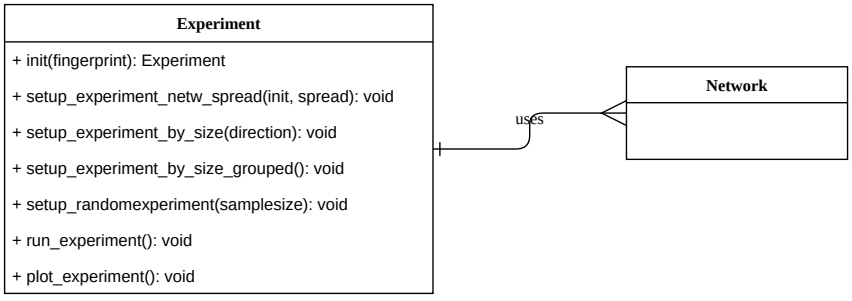


Figure 4.2: Class relationship between Experiment and Network.

4.1.4 Command line interface

To simplify the interaction with the simulation software to facilitate execution of different experiment parameters without the need of changing the source code a simple command line interface can be used. The file is called `simulate.py` and uses the package **docopt** to create the interface and the parser. The interface is defines as shown in listing 2. While `parse` starts the preprocessing from a channel file all other commands execute different types of experiments and take respective parameters. The optional `--charts` flag skips the experiment and only generates the resulting charts.

```
Usage:
  simulate.py parse <filename>
  simulate.py randomexperiment <fingerprint> <samplesize> [--charts]
  simulate.py bysize <fingerprint> [(asc|desc)] [--charts]
  simulate.py groupedsize <fingerprint> [--charts]
  simulate.py spread <fingerprint> <init> <spread> [--charts]
  simulate.py -h | --help
  simulate.py --version

Options:
  -h --help     Show this screen.
  --version     Show version.
  --charts      Skips the experiment and creates only the charts.
```

Listing 2: Description of the command line interface

4.2 Data collection

Data for the simulation was first collected from a Lightning node. This section describes the data structure obtained and what features were selected as input for the next step, the preprocessing.

In order to run a simulation that resembles the real network closely, data from a productive Lightning node are best fitted to construct the model. Lightning is an open network which means it is easy to setup a node and download all the channel information. There are three major implementations of Lightning which technically could deliver the data. The implementation C-Lightning offers a flexible interface to write own plugins which can interact with the node over Remote Procedure Calls . I wrote a plugin [3] in Python that writes all known channels to a JSON file. The code snippet 3 shows the raw data which is returned by C-Lightning.

rpc in glossary

[3] https://github.com/TKone7/clightning-plugins/tree/3b47ab992413d7bf9747be1320654b3478c60d37/dumpgraph

```
{
   "channels": [
      {
         "source": "02ad6fb8d693dc1e4569bcedefadf5f72a931ae027dc0f0c544
         ↪  b34c1c6f3b9a02b",
         "destination": "03fab7f8655169ea77d9691d4bd359e97782cb6177a6f7
         ↪  6383994ed9c262af97a5",
         "short_channel_id": "513909x1248x0",
         "public": true,
         "satoshis": 20000,
         "amount_msat": "20000000msat",
         "message_flags": 1,
         "channel_flags": 0,
         "active": true,
         "last_update": 1594283842,
         "base_fee_millisatoshi": 0,
         "fee_per_millionth": 1,
         "delay": 144,
         "htlc_minimum_msat": "1000msat",
         "htlc_maximum_msat": "20000000msat"
      },
      ...
   ]
}
```

Listing 3: Raw channels provided by C-Lightning.

The raw data contains one key named **channels** that consists of an array. At the time of data extraction this array contained 58934 objects. Nodes can open new channels and close existing ones at any point in time. The state of the network is therefore not static. All experiments conducted in this thesis are based on the local view of my node at 30.07.2020. Each object represents one directed edge. Keys **source** and **destination** refer to the involved node's public key. This public key is used as a pseudonym to preserve the node operator's privacy. Furthermore, private public key pairs are used to encrypt (see onion routing 1.2.3) and sign messages. Since one entry only represents one directed edge there shoud be a second entry with the same `short_channel_id` with reversed source destination nodes to make the bidirectional channel complete. Field `sathoshis` tells the total channel capacity in the unit "sathoshi" . The field `base_fee_millisatoshi` is the fixed fee each payment has to pay when routing through the channel. The variable fee to pay for each million satoshis is defined by `fee_per_millionth`. All ofther fields are not important for the simulation and will be ignored. The plugin extracts the highlighted parts and writes them into a file as shown in listing 4.

define units

Listing 4 shows two entries of the same `short_channel_id` which means they represent the same channel but in opposite directions. Source and destination of both entries confirm this and also the capacity does match. It is also an example for a channel with different fee policies in each direction. For example a payment of 200000 satoshis routed from **02ad6f** to **03fab7** would charge no fixed fee but $\frac{200000}{1000000} * 1 = 0.2$ sathoshi in variable fee. The same payment routed in the opposite direction would charge the same variable fee but additionally 1000 millisatoshi

(equals 1 satoshi) in fixed fee. So a total of 1,2 satoshi. The complete `channels.json` file can be downloaded from the Github repository.

```
{
    "channels": [
        {
            "source": "02ad6fb8d693dc1e4569bcedefadf5f72a931ae027dc0f0c
            ↪ 544b34c1c6f3b9a02b",
            "destination": "03fab7f8655169ea77d9691d4bd359e97782cb6177a
            ↪ 6f76383994ed9c262af97a5",
            "short_channel_id": "513909x1248x0",
            "satoshis": 20000,
            "base_fee_millisatoshi": 0,
            "fee_per_millionth": 1
        },
        {
            "source": "03fab7f8655169ea77d9691d4bd359e97782cb6177a6f763
            ↪ 83994ed9c262af97a5",
            "destination": "02ad6fb8d693dc1e4569bcedefadf5f72a931ae027d
            ↪ c0f0c544b34c1c6f3b9a02b",
            "short_channel_id": "513909x1248x0",
            "satoshis": 20000,
            "base_fee_millisatoshi": 1000,
            "fee_per_millionth": 1
        },
    ...
    ]
}
```

Listing 4: Sample channels file produced by the plugin.

4.3 Preprocessing

The date obtained in section 4.2 is not yet ready to be used to construct a network. There exist channels in the dataset for which only one of the two edges is known. Furthermore the information about the distribution of the channel balance is unknown but must be allocated somehow. These two issues are being addressed in the preprocessing which only has to be executed once. The result from the preprocessing is a `networkx.DiGraph` which is then passed to the constructor of the Network. It is then accessible through the attribute `self.G` of the Network object.

4.3.1 Remove incomplete data

The data sample presented in listing 4 shows that each Lightning channel is represented by two entries with opposite source-destination node pairs. In the obtained dataset not every entry has its matching counterpart. The experiments should only deal with channels that can be used in both direction (normal case), therefore, these data objects are simply deleted from the dataset. This reduced the dataset from a total of 58934 to 46366 (-6172) data objects, representing $frac463662 = 23183$ channels.

I use a **simple directed graph** to represent the Lightning network. Multiple edges with the same source and destination node are not permitted. The Lightning protocol does not prevent nodes to open multiple channels with the same partner. Therefore, multiple channels are reduced to one single channel.

4.3.2 Allocate local balances

The public data only contains information about the total channel capacity, the local distribution between the two channel partners is unknown. To run a simulation some distribution must be assumed. As all channels are single funded this means the capacity was all on one side initially. The best guess to distribute channel balances is therefore to assume all channels were just opened and therefore only one party hold the full capacity in its balance. However, there is no hint on who opened the channel. This is why the preprocessing allocates 100% of the capacity to one of the parties **randomly**. To ensure reproducibility of the preprocessing the allocation of the balance should be random but deterministic. Meaning the same channel distributes the capacity to the same node in consecutive runs. Based of the evenness of the result of a hash function the node holding the balance is selected (see listing 5).

explain hash function

```
1  # shuffle source<->destination randomly
2  shuffled = set()
3  for channel in reduced:
4      input = bytearray(channel[0] + channel[1], 'utf-8')
5      hash_object = hashlib.sha256(input)
6      if hash_object.digest()[0] % 2 == 0:
7          shuffled.add(tuple([channel[1], channel[0]]))
8      else:
9          shuffled.add(channel)
```

Listing 5: Random allocation of channel balance.

Depending on the allocation of the balances some nodes are not able to rebalance payments with the rest of the network. To find a network in which each node can reach each other node a NetworkX graph is created with only **one** edge pointing from the node with balance to the node without balance. From this graph the **strongly connected component** is selected. This is the directed graph in which a sequence of edges form a path from every vertex to every other vertex (Even & Tarjan, 1975). The resulting strongly connected component consists of 21147 channels (-2036) and has a diameter of 8. This means the longest path between any two nodes is 8.

4.4 Simulate rebalancing

In section 3.2.2 we defined the rebalancing algorithm each node following the protocol change would implement. This section demonstrates how it is implemented in the simulation. Figure 4.3 serves as a big picture how such a rebalancing is simulated. The next sections will go into more detail.

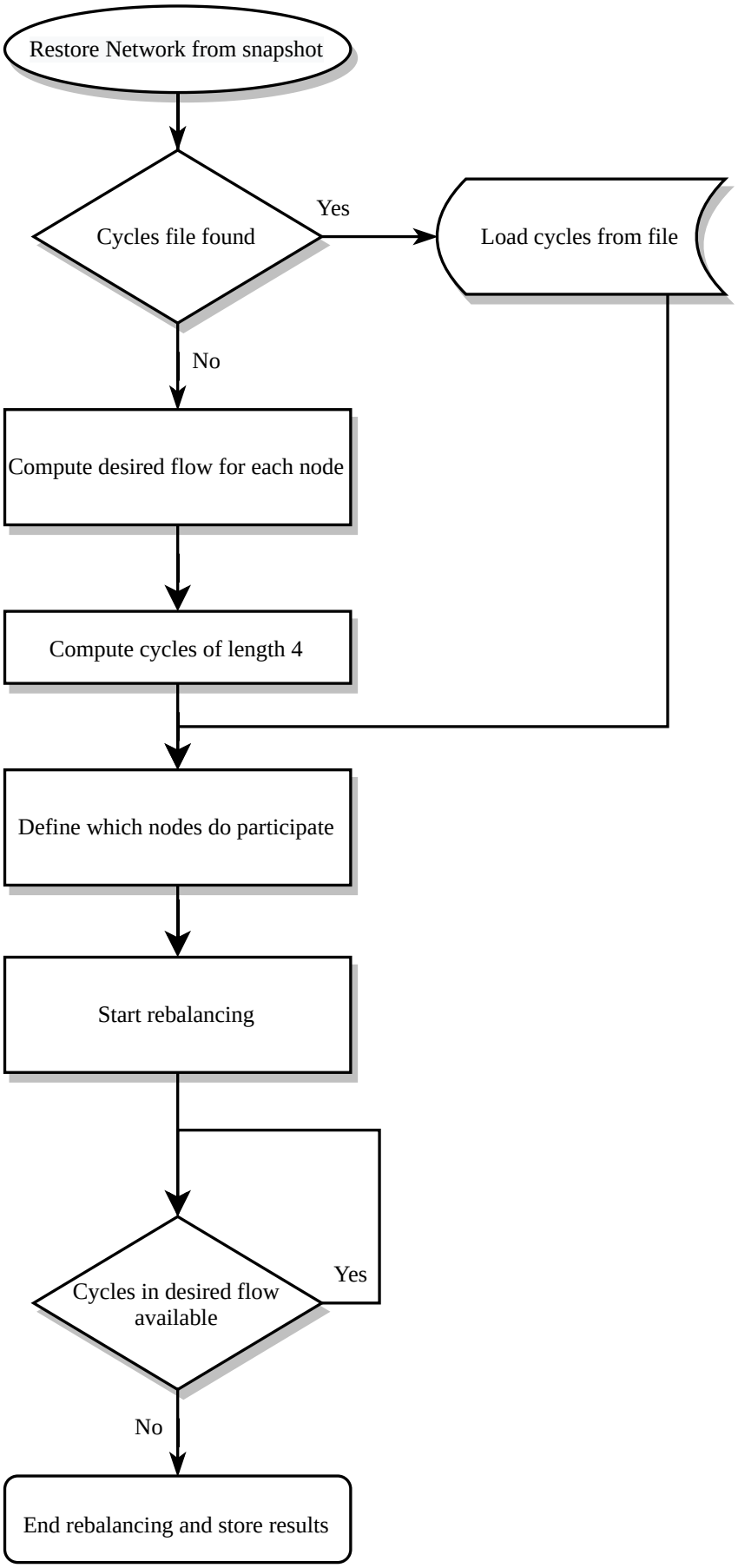


Figure 4.3: Flow chart of a rebalancing experiment.

4.4.1 Determine the desired flow

While the rebalancing in reality would require each node to calculate the desired transactions individually we assume that rebalancing hints are shared among nodes and we can take a holistic view of the different flows each node would want to make. The method in listing 6 covers steps 1 to 4 and part of step 5 described in section "Proposed algorithm".

The method `__compute_rebalance_network(self)` creates a new NetworkX graph which is derived from the original network graph (stored in `self.G`). Line 4 starts a for-loop which iterates over all edges (u, v) and calculates how much node u would like to send out over this edge. The first step is to calculate the node's **node balance coefficient**. This value is already pre-calculated and can be retrieved from the attributes stored in the NetworkX graph (line 7). Lines 8-10 retrieves the balance and capacity of the edge and calculates the **channel balance coefficient**. The formula from step 5 is used to calculate the amount $a = c(e) \cdot (\zeta_{(u,v)} - \nu_u)$ node u would want to rebalance (line 11). This calculation returns a positive integer which represents the need to send out this amount while a negative integer represents the wish to receive this amount.

Since a payment channel is always represented by two edges there are two nodes that have some different need to rebalance it. As the protocol foresees only rebalance transactions from which both benefit, an agreement must be found. In the case both nodes would want to rebalance into different directions (both want to send or receive at the same time) the channel is completely removed from the flow graph. As in the replicated network all channel balance lies on one end of the channel this will not be the case. However, most likely one of the nodes wants to send more than the counterparty wants to receive (or the other way around). This case is handled in lines 12-19. The lower of the two values is taken and stored into the flow graph with respect to the sign.

The result of this procedure is a property `self.flow` which contains a network graph that has two opposite directed edges for each channel that wants to be rebalanced by both parties. The amount stored in the attribute `liquidity` is the lower value the two channel partners could agree on. The value is the same for edges of a channel, but positive on the edge that wants to send and negative on the one that wants to receive. This flow graph will be used later to compute possible rebalancing paths.

```
def __compute_rebalance_network(self):
        self.flow = nx.DiGraph()
        delete_edges = []
        for u, v in self.G.edges():
            if u in self.__excluded or v in self.__excluded:
                continue
            nbc = self.G.nodes[u]['nbc']
            balance = self.G[u][v]['balance']
            capacity = self.G[u][v]['capacity']
            cbc = balance / capacity
            amt = int(capacity * (cbc - nbc))
            if (v, u) in self.flow.edges():
                amt_cp = self.flow[v][u]['liquidity']
                if np.sign(amt) == np.sign(amt_cp):
                    ...
                common = min(abs(amt), abs(amt_cp))
                amt_cp = common * np.sign(amt_cp)
                amt = common * np.sign(amt)
                self.flow[v][u]['liquidity'] = amt_cp
            self.flow.add_edge(u, v, liquidity=amt)
```

Listing 6: Method to calculate each nodes willingness to rebalance.

4.4.2 Compute rebalancing cycle

Step 5 of the rebalancing algorithm tells a node u to construct circular payments using a path $p = [v, x_1, \ldots, x_n, u]$ where $\zeta_{(u,v)} > \nu_u$ and $\zeta_{(x_n,u)} < \nu_u$. Such a rebalancing transaction helps a node to adjust both channel balance coefficients to the node balance coefficient. In the real network nodes would individually calculate those paths based on the **rebalancing hints** they receive from their local neighbourhood. In the simulation we already have this holistic view since we calculated the desired flow in section 4.4.1.

Before we extract the rebalancing cycles from the flow network we define the local neighbourhood of a node as the friend of a friend network. This is the network within which the proposed protocol foresees nodes to share hints about their willingness to rebalance. If we assume node `self` in figure 4.4 has only two channels open with `Friend1` and `Friend2`. These friends would both provide information about all their channel balances (marked as green edges). This allows the node `self` to assess the rebalancing path $p = [self, Friend1, D, Friend2, self]$ with more insight about the willingness of the nodes to participate. However, this restricts the paths for rebalancing to the ones of length 4 or less. Although node `self` also knows about the channel $e = (c, x)$, it has no insight about their imbalance since it is not one of its friends channel. Hence, this path of length 5 is not considered for rebalancing.

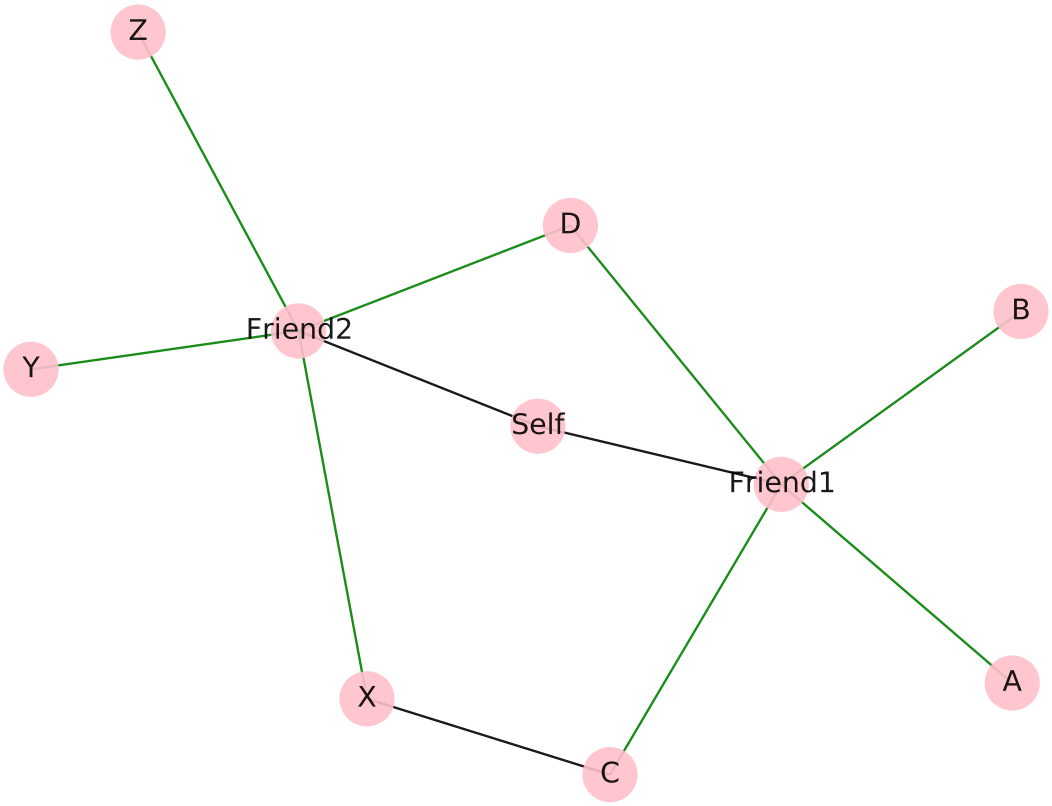


Figure 4.4: Sharing rebalance hints in the friend-of-a-friend network.

Since the flow network is already calculated, we can simply extract all cycles of length 4 or less as shown in the method `compute_circles()` in listing 7. Line 4-6 create an extract of the flow network with only positive edges. The negative edges are just the counterparts of same size in the opposite direction and can be deleted. Lines 7-9 then iterate over each edge $e = (u, v)$ and uses the built-in function `networkx.all_simple_paths()` to find all paths from v to u. This results in paths of length 3 or less and the edge $e = (u, v)$ must be added later to make it a full circle. Since node u is always the last node in the found path it does not have to be stored at this stage. All found cycles are then randomly ordered in a deterministic manner and stored to a file. During the future rebalancing process those set of circular paths will not change as we do not allow channels to become less balanced. This means the liquidity in the flow network only decreases and edges might disappear completely, but there is no possibility for new edges to appear. This fact is very helpful as the computation of the paths needs to be done only once per network. If during the restore procedure an existing file of exported paths is found, this will be automatically imported.

```
    def compute_circles(self, force=False):
        ...
        cycles4 = []
        pos_edges = [e for e in self.flow.edges(data=True) if
        ↪  e[2]['liquidity'] > 0]
        pos_flow = nx.DiGraph()
        pos_flow.add_edges_from(pos_edges)
        for i, (u, v) in enumerate(pos_flow.edges):
            paths = [p for p in nx.all_simple_paths(pos_flow, v, u, 3)]
            [cycles4.append(p) for p in paths if len(p) <= 4]
        ...
        self.__cycles4 = cycles4
        random.seed(10)
        self.__cycles4.sort()
        random.shuffle(self.__cycles4)
        self.__store_cycles()
```

Listing 7: Extracts cycles of length 4 (or less) from the flow graph.

4.5 Performance Measures

list the two existing and the third new performance measurement

4.5.1 Success rate on shortest path

bla

4.5.2 Payment size on shortest path

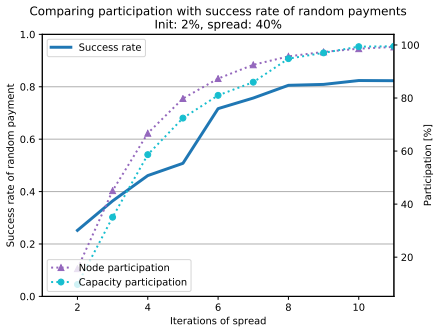
dl

4.5.3 Ability to route different payment sizes

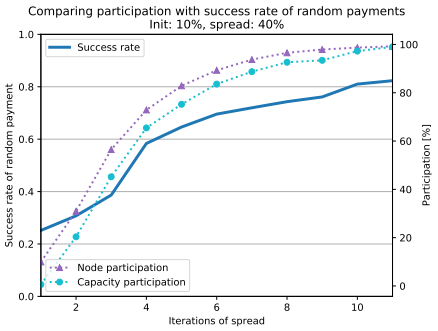
bla

5 Results

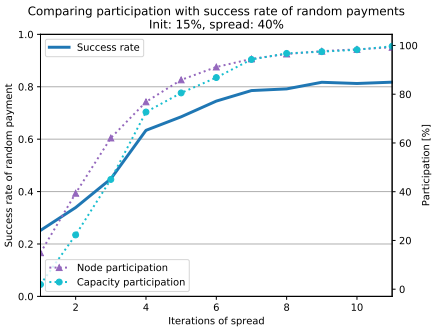
Show a lot of charts :-)



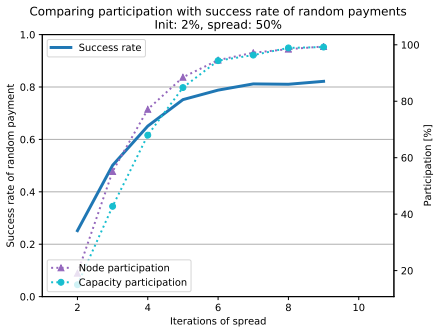
(a) Init: 2 Spread: 40



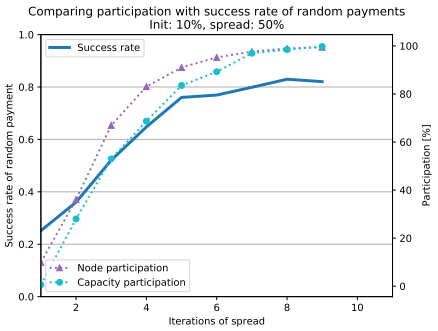
(b) Init: 10 Spread: 40



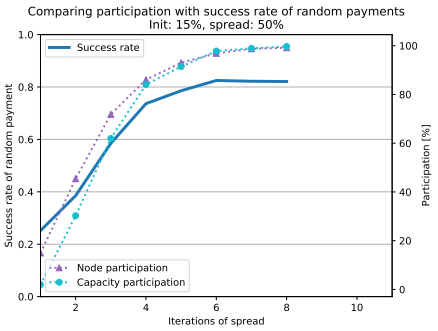
(c) Init: 15 Spread: 40



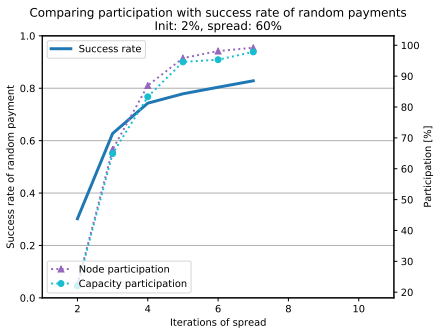
(d) Init: 2 Spread: 50



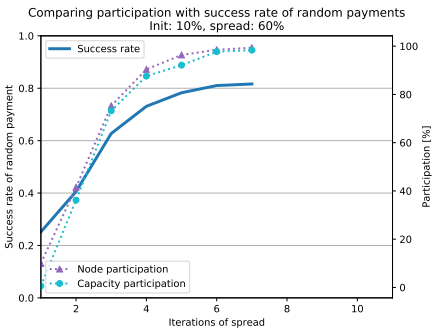
(e) Init: 10 Spread: 50



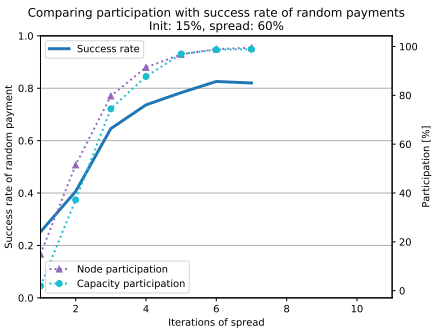
(f) Init: 15 Spread: 50



(g) Init: 2 Spread: 60

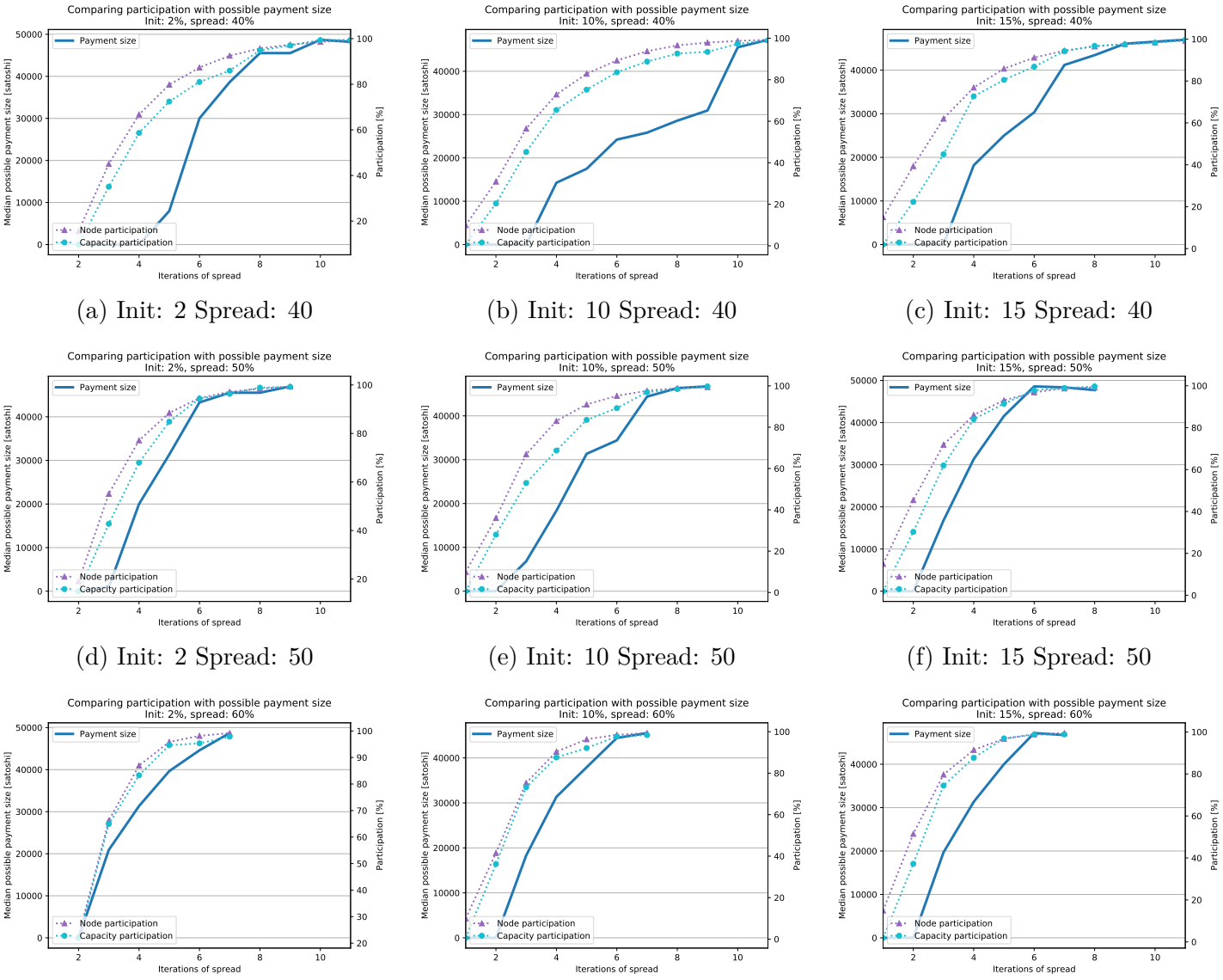


(h) Init: 10 Spread: 60



(i) Init: 15 Spread: 60

Figure 5.1: Success Rate

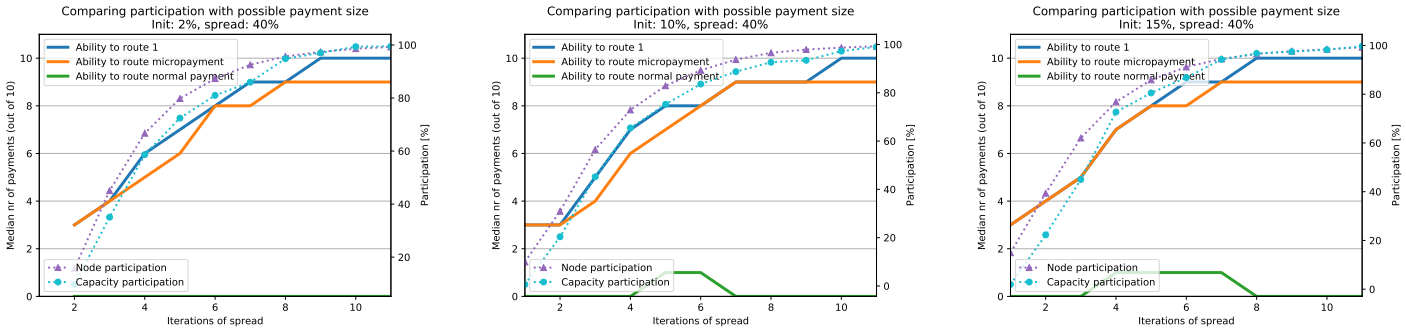


(a) Init: 2 Spread: 40 (b) Init: 10 Spread: 40 (c) Init: 15 Spread: 40

(d) Init: 2 Spread: 50 (e) Init: 10 Spread: 50 (f) Init: 15 Spread: 50

(g) Init: 2 Spread: 60 (h) Init: 10 Spread: 60 (i) Init: 15 Spread: 60

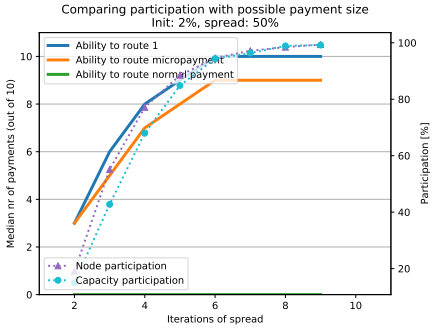
Figure 5.2: Median possible payment size



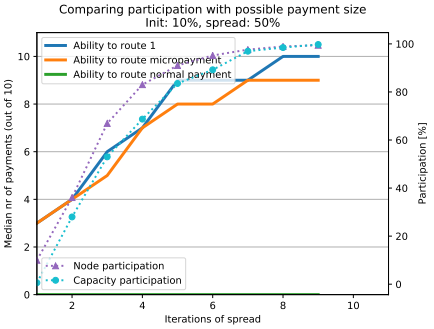
(a) Init: 2 Spread: 40

(b) Init: 10 Spread: 40

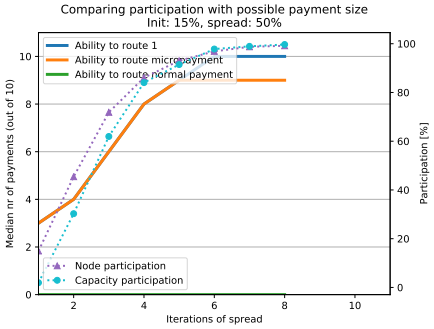
(c) Init: 15 Spread: 40



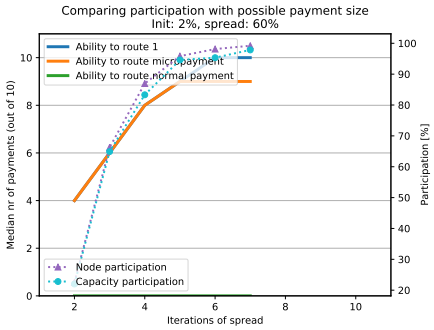
(d) Init: 2 Spread: 50



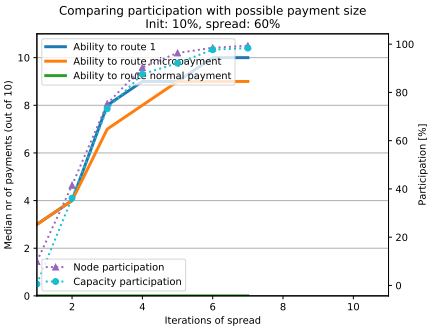
(e) Init: 10 Spread: 50



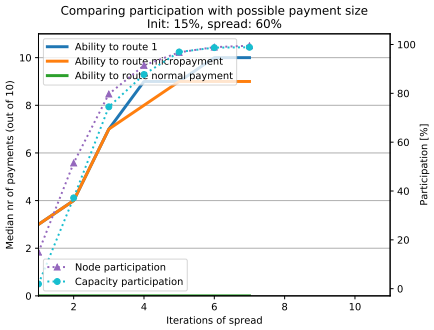
(f) Init: 15 Spread: 50



(g) Init: 2 Spread: 60

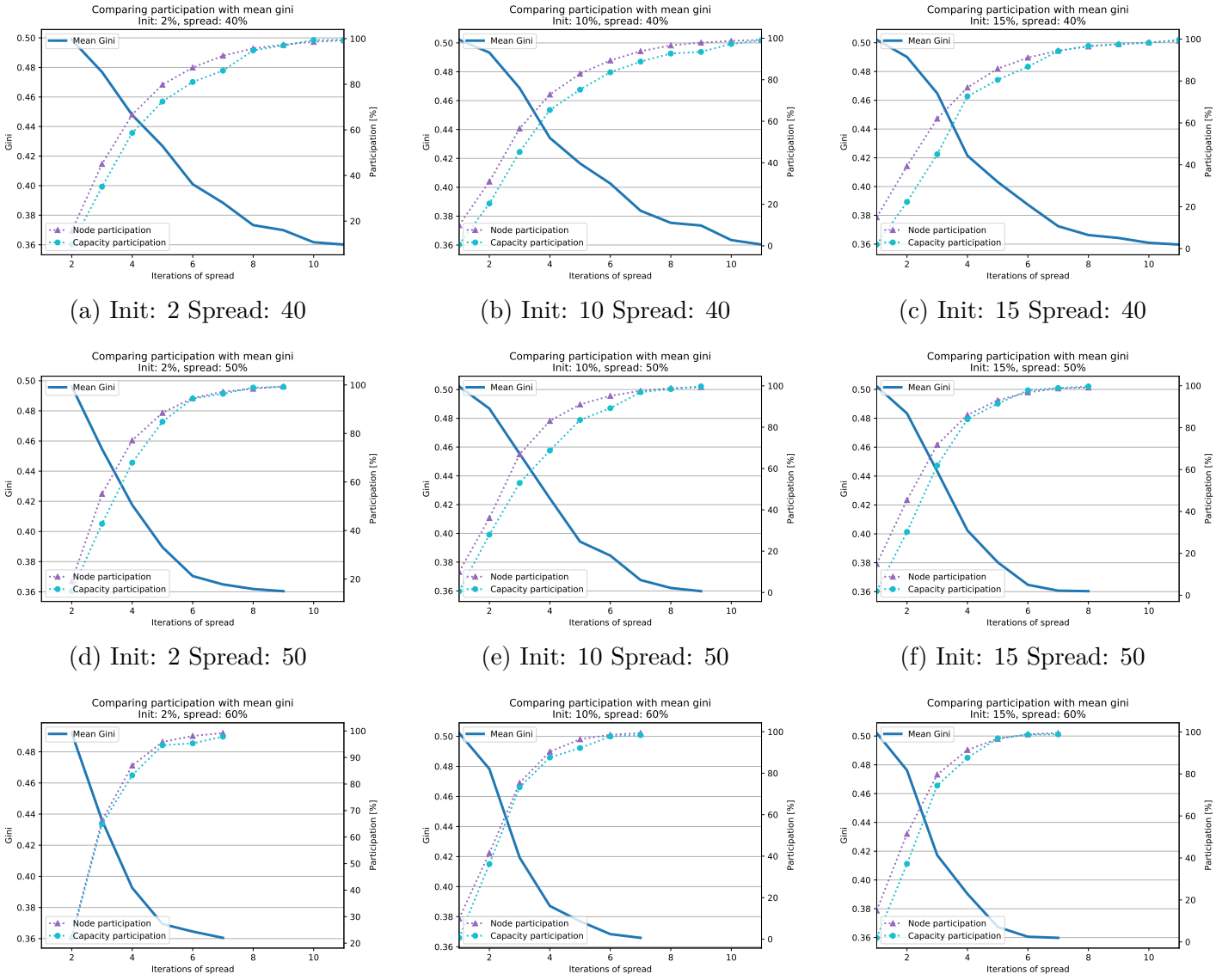


(h) Init: 10 Spread: 60



(i) Init: 15 Spread: 60

Figure 5.3: Different payment sizes



(a) Init: 2 Spread: 40 (b) Init: 10 Spread: 40 (c) Init: 15 Spread: 40

(d) Init: 2 Spread: 50 (e) Init: 10 Spread: 50 (f) Init: 15 Spread: 50

(g) Init: 2 Spread: 60 (h) Init: 10 Spread: 60 (i) Init: 15 Spread: 60

Figure 5.4: Different payment sizes

6 Conclusion & Outlook

References

Back, A. (2002, August 1). *Hashcash - a denial of service counter-measure*. Retrieved from http://cypherspace.org/hashcash/hashcash.pdf

blocks-size. (n.d.). Retrieved July 6, 2020, from https://www.blockchain.com/charts/blocks-size

Chaum, D. (1983). Blind signatures for untraceable payments. In D. Chaum, R. L. Rivest, & A. T. Sherman (Eds.), *Advances in cryptology* (pp. 199–203). doi:10.1007/978-1-4757-0602-4_18

Even, S., & Tarjan, R. E. (1975). Network flow and testing graph connectivity. *SIAM Journal on Computing*, *4*(4), 507–518. doi:10.1137/0204043

Haber, S., & Stornetta, W. S. (1991). How to time-stamp a digital document. *Journal of Cryptology*, *3*(2), 99–111. doi:10.1007/BF00196791

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system, 9. Retrieved June 7, 2020, from https://bitcoin.org/bitcoin.pdf

Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. (2016). *Bitcoin and cryptocurrency technologies*.

Pickhardt, R., & Nowostawski, M. (2019). *Imbalance measure and proactive channel rebalancing algorithm for the Lightning Network*.

Poon, J., & Dryja, T. (2016). The bitcoin lightning network: Scalable off-chain instant payments, 59.

Rosen, K. H. (2012). *Discrete mathematics and its applications* (7th ed). New York: McGraw-Hill.

Al-Taie, M. Z., & Kadry, S. (2017). *Python for graph and network analysis*. OCLC: 978274331. Retrieved July 11, 2020, from https://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=1488266

Tikhomirov, S., Pickhardt, R., Biryukov, A., & Nowostawski, M. (2020). Probing Channel Balances in the Lightning Network. *arXiv:2004.00333 [cs]*. version: 1. arXiv: 2004.00333. Retrieved July 13, 2020, from http://arxiv.org/abs/2004.00333

Glossary

Glossary

double spend problem Problem in digital cash systems that a digital token can be duplicated at will and used as payment to multiple receivers at the same time making it difficult to detect the fraud.. 1

A Some appendix

maybe...