Capabilities and Limitations of Payment Channel Networks for Blockchain Scalability

Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (31^{st} cycle)

Capabilities and Limitations of Payment Channel Networks for Blockchain Scalability

Marco Conoscenti
* * * * * *

Supervisor
Prof. Juan Carlos De Martin

Politecnico di Torino
2019

I hereby declare that the contents and organisation of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

.....

Marco Conoscenti

Turin, 2019

Summary

Bitcoin and other blockchain-based cryptocurrency do not scale, because the blockchain limits transaction throughput. Payment channel networks are the most promising solution to address scalability, as they enable off-blockchain payments that are not subject to the blockchain throughput limit. The Lightning Network (LN) is the mainstream payment channel network, built on top of the Bitcoin blockchain. The LN leverages the Hashed Timelock Contract (HTLC) to transfer off-chain payments in a trustless and secure way. However, the Lightning Network and HTLC-based payment channel networks present critical features that might undermine their correct functioning and therefore need to be investigated. Some examples of these critical features are channel economic capacity, which limits payment amounts, channel unbalancing, which makes payment channels unusable in one direction, and uncooperative behavior of nodes, which causes increased payment time and failures.

This work presents CLoTH, a simulator for HTLC payment networks. CLoTH is an original payment network simulator, as it constitutes a precise mapping of the LN code functions. As input, it takes parameters defining a payment channel network (e.g., number of channels per node, average channel capacity) and parameters defining payments (e.g., payment amounts and payment rate). It simulates the input-defined payments on the input-defined HTLC payment network. It produces performance measures in terms of payment-related statistics, such as probability of payment failure and time to complete payments. CLoTH is a valuable tool to identify issues, analyze solutions and steer future developments of payment channel networks.

This work discusses three groups of simulations conducted using the CLoTH simulator. In the first group a snapshot of the Lightning Network was given as input to the simulator. The goal of these simulations was to find configurations in which a payment is more likely to fail than to succeed in the LN. Simulations of the second group were conducted on synthetic networks, i.e., payment networks generated by the simulator using their statistical description. The goal of these simulations was to study the impact of each simulator input parameter on payment network performance. The third group of simulations aimed to analyze the effects of network and protocol modifications in the Lightning Network, like for instance

the removal of network hubs and channel rebalancing approaches.

Simulation results prove that the current most relevant issues of the Lightning Network are limited channel capacities and channel unbalancing, which both cause payment failures. A rebalancing approach designed and simulated in this work effectively addresses channel unbalancing. At the same time, the simulations performed prove that the Lightning Network is resilient to the removal of the most connected network hubs and can support a contained level of node uncooperativeness.

This work is one of the first attempts to thoroughly examine capabilities and limitations of payment channel networks, and to provide viable solutions. The CLoTH simulator is a valid and useful tool for systematically studying payment channel networks and assisting their development.

Acknowledgements

The author wishes to thank his supervisor, Prof. Juan Carlos De Martin, who gave him the opportunity to do the PhD, in a place where critical thinking is continuously stimulated. The author thanks Dr. Antonio Vetrò for his advice and help in all the three years of the PhD. The author wishes to express sincere gratitude to Dr. Federico Spini for his insightful guidance and fundamental collaboration on the research topics discussed in this work. Finally, special thanks go to all the colleagues at the Nexa Center for their support and affection. They are (in random order): Selina Fenoglietto, Giuseppe Futia, Giovanni Garifo, Pasquale Pellegrino, Elena Beretta, Francesco Ruggiero, Mattia Plazio, Antonio Santangelo, Simone Basso and Lorenzo Canova.

"We can forgive a man for making a useful thing as long as he does not admire it."

Oscar Wilde

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Bitcoin is a decentralized cryptocurrency that allows mistrusting peers to send and receive monetary value without the need for intermediaries [29]. Bitcoin relies on the blockchain, a distributed peer-to-peer public ledger which stores all the history of Bitcoin economic transactions. The replication of the ledger and the synchronization mechanism of the replicas entail a capped transaction throughput [38], which prevents cryptocurrencies from scaling and from becoming world-wide payment systems.

Payment channels are the most explored technical proposals that address the issue of blockchain scalability [33, 6, 28, 35, 3, 20, 34]. They enable off-chain payments, i.e., payments that do not need to be registered on the blockchain and thus are not subject to the blockchain throughput limit. A payment channel is a two-party ledger which is updated off-chain: the two involved parties, the channel endpoints, can bidirectionally exchange an unbounded number of off-chain payments through the channel. The blockchain is only used to open and close channels, and it does not register the payments that take place in the channels.

Two-party payment channels can be linked together to build a *payment channel network*. This allows parties not directly connected by a payment channel to send/receive off-chain payments which are routed across a network of linked payment channels.

The *Lightning Network* (LN) [33] is the most developed and mainstream payment channel network, built for Bitcoin. In March 2018, the first beta version of one of the implementations of the LN protocol was released. Since than, the Lightning Network can be used with bitcoins of real value. At the time of writing, the LN is constituted by more than 3 thousands nodes and almost 25 thousands payment channels, with around 660 bitcoins in the network (being worth more than 3 millions of dollars).

The LN protocol leverages a particular contract called *Hashed Timelock Contract* (HTLC). The HTLC allows the transfer of off-chain payments through multiple payment channels in a trustless way. Henceforth, in this work the *HTLC*

payment network is defined as a payment channel network where payments are transferred using HTLC contracts.

At their current state of development, the Lightning Network and payment channel networks are characterised by features that, if not properly understood, implemented and controlled, may undermine the development of a healthy payment network that supports fast and successful payments. Some examples of these critical features that need to be studied are the following: (i) routing, i.e., a good routing path that does not cause a payment failure depends on the knowledge of an up-to-date network topology; (ii) channel economic capacity, which limits payment amounts; (iii) channel unbalancing, namely, the condition of a skewed channel that has one endpoint depleted due to a number of unidirectional payments, an issue not internally addressed by the LN protocol; (iv) uncooperative behavior of nodes involved in a payment route, which causes lock of funds in transfer and increased payment time.

In this work CLoTH was developed, a simulator for HTLC payment networks. The originality of CLoTH is that it constitutes a precise and complete mapping of the LN code functions which implement an HTLC payment network. The simulator allows the identification of issues of HTLC payment networks and the estimation of optimization actions before deploying them. CLoTH is a valuable tool for systematically analyzing payment channel networks and for steering their future developments.

The CLoTH simulator is a discrete-event simulator that simulates payments on HTLC payment networks. As input, it takes parameters defining an HTLC network (e.g., number of nodes and number of channels per node) and parameters of the payments to be simulated on the defined HTLC network (e.g., payment rate and payment amounts). As output, it generates performance measures in the form of payment statistics, such as the probability of payment failure and the mean payment complete time.

Some examples of questions that can be answered by CLoTH are: “How many channels per node are required to generate a well connected network?”; “How do uncooperative nodes influence payment time?”; “How does payment rate influence the chance of payment failure?”; “How is performance affected by adding a node with a specific set of payment channels in a specific section of the network?”.

The general purpose of this work is to systematically analyze capabilities and limitations of payment channel networks using the CLoTH simulator. Three research questions are addressed in this thesis.

RQ1 *Which are the non-operative cases of the LN mainnet?*

The LN mainnet is the network of channels and nodes running the LN protocol. The term *mainnet* indicates the main network in the cryptocurrency jargon, to

differentiate it from the test network. Here, the LN mainnet is defined as non-operative when a payment is more likely to fail than to succeed.

To answer this research question, a set of simulations were conducted in which CLoTH took a snapshot of the LN mainnet as input. These simulations searched for configurations of input parameters causing non-operative cases. Multiple non-operative cases were found, which were mainly due to insufficient channel capacities and to channel unbalancing.

RQ2 *Which is the impact of the simulator input parameters on performance of payment channel networks?*

To answer this research question, multiple simulations were run on synthetic networks, i.e., networks generated by CLoTH using the simulator input parameters. In general, simulation results prove that the synthetic networks analyzed have a good performance. For instance, they can support a contained level of node uncooperativeness.

RQ3 *How do network and protocol modifications affect performance of the LN mainnet?*

To answer this research question, simulations were conducted on the LN mainnet when different protocol and network modifications were applied.

With regard to the protocol modifications, rebalancing approaches were implemented in the simulator to tackle the issue of channel unbalancing. Simulation results prove that one of the approaches analyzed reduces by one fourth the probability of payment failure for unbalanced channels with respect to the case in which no rebalancing approach is implemented.

Two separate network modifications were analyzed. The first consisted in removing hubs (i.e., the most connected nodes) from the LN mainnet. Simulations conducted on the resulting networks without hubs prove that the LN mainnet is resilient to hub removal. In the second network modification, classes of nodes (payees only, payers only and hybrid payee-payer nodes) were defined in order to simulate a typical use case of the Lightning Network, in which most of the payments are directed to a few service-provider nodes. Simulation results show that channels directing to the service providers become unbalanced, thus causing a significant probability of payment failures for unbalancing.

The remainder of this work is organized as follows. Chapter 2 provides the background on Bitcoin, blokchain and payment channel networks. In Chapter 3 the CLoTH simulator is described in detail. Chapter 4 discusses the first group of simulations conducted on the LN mainnet to answer Research Question 1. In

Chapter 5 the simulations on synthetic networks are described and Research Question 2 is answered. Chapter 6 presents the third group of simulations on network and protocol modifications and Research Question 3 is answered. Finally, Chapter 7 provides conclusions and delineates the future work.

Chapter 2

Background on Bitcoin and Payment Channel Networks

This Chapter provides the background on Bitcoin and the Lightning Network required to understand the rest of the thesis. It is organized as follows. Section 2.1 is on Bitcoin and the blockchain. Section 2.2 discusses the issue of scalability and possible solutions. Finally, Section 2.3 describes payment channel networks and especially the Lightning Network, including also a brief literature on such networks.

2.1 Bitcoin and the Blockchain

This Section provides the background on Bitcoin and the blockchain. It is not meant to be complete and exhaustive. Readers are referred to [30, 1] for detailed explanations of the system.

Bitcoin cryptography and transactions Bitcoin is a digital payment system and digital money. It is implemented by a free software, which anyone is free to download and use. A computer running the Bitcoin software is called *Bitcoin node*, it is connected to the other Bitcoin nodes and it is part of the Bitcoin peer-to-peer network.

Bitcoin makes use of asymmetric cryptography. Each node has a public and a corresponding private key[1]. The public key serves as a pseudonymous identity of the node and as an address where the node receives the Bitcoin currency (whose unit is called *bitcoin*, symbol BTC). The corresponding private key allows the node to prove ownership of the received bitcoins and to spend them.

[1]In practice, for unlinkability and privacy purposes it is recommendable that each node has more than one pair of public and private keys.

A bitcoin *transaction* allows parties to exchange bitcoins. A transaction is made of inputs and outputs. An output contains the bitcoins transferred and a locking script, which codes the conditions necessary to spend the bitcoins contained in the output. An input is a reference to a previous output and an unlocking script, which satisfies the spending conditions and spends the bitcoins contained in the referenced output.

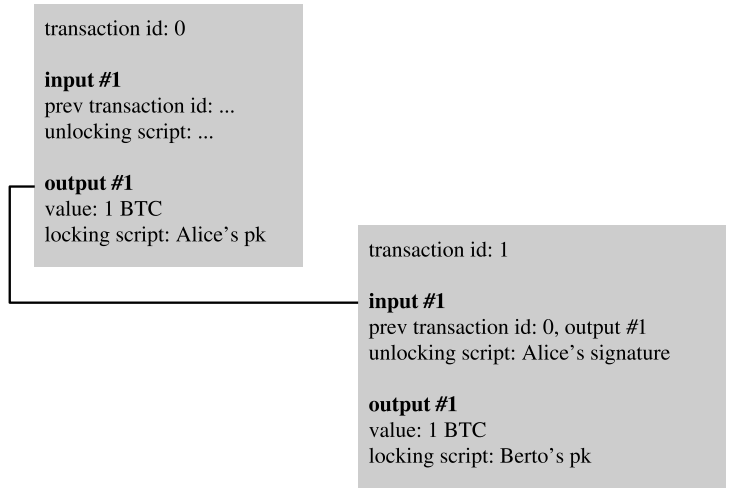


Figure 2.1: Bitcoin transactions.

For example, Figure 2.1 shows a transaction in which Alice transfers one bitcoin to Berto. The input is a reference to a previous transaction output containing one bitcoin earlier sent to Alice and the digital signature of Alice, necessary to spend that bitcoin. The output is made of one bitcoin and a locking script, which contains the public key of Berto.

It is possible to perform more elaborate transactions than simple transfer of bitcoins from one party to another, by coding complex locking scripts. An example is the *multisignature* script, which contains N public keys and requires M out of those N corresponding private key signatures to spend the output locked by the script.

Double-spending problem The main problem affecting digital money is double-spending, namely, the fact that the very same coin can be spent twice. In fact, digital money can be copied without effort, as any other digital object.

Ecash [4], introduced by David Chaum in 1983, was the first digital payment system that solved the double spending problem. Chaum invented a scheme where all spent digital notes, identified by a unique serial number, are registered on a ledger. In that way it is easy to check against the ledger whether a certain note has already been spent.

Ecash is a centralized payment system. A trusted central entity, like a bank, is required to maintain the ledger and check for double-spending.

Also in Bitcoin the double-spending is solved by storing transactions in a ledger, called *blockchain*. However, differently from Ecash, the Bitcoin blockchain is not managed by a centralized entity, instead by all Bitcoin nodes. For this reason, Bitcoin is the first payment system which achieves decentralization: it operates without the need for a trusted central entity.

The blockchain The blockchain is the ledger storing all occurred Bitcoin transactions. It is called "blockchain" because its structure is a chain of blocks, as Figure 2.2 shows. A block is a group of transactions. Each block has a reference to the previous block, thus creating a chain. In particular, each block contains the hash to the previous block. Such structure makes the blockchain tamper-proof: if a transaction in a block is modified, the block hash becomes invalid, and also all the hashes of the blocks following the block of the modified transaction.

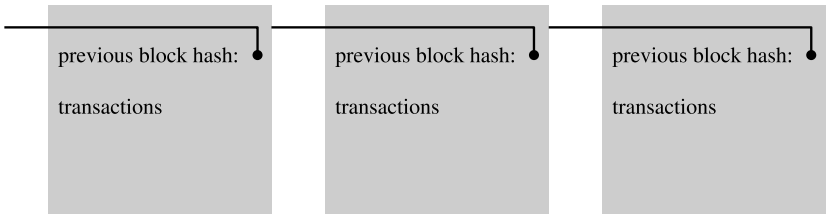


Figure 2.2: The Bitcoin blockchain.

The blockchain is distributed: several Bitcoin nodes maintain a copy of it. To keep consistent the replicas, a *distributed consensus protocol* is necessary. Having multiple replicas managed by multiple nodes, a distributed consensus protocol ensures that all the replicas remain consistent among them and store correct values, even in presence of faulty or malicious nodes.

All distributed consensus protocols preceding Bitcoin work only under a specific assumption. In particular, they work only if, having N nodes in the distributed system, at maximum a certain fraction (e.g. one third) of N is malicious/faulty [23]. As a consequence, all those distributed consensus protocols are vulnerable to the Sybil attack. In that attack, the attacker can create any number of nodes which seem different but are all under its control. The attacker can thus undermine the distributed consensus, by creating more malicious nodes that the ones tolerated by the protocol. The only solution is to count and identify participating nodes, which are trusted and not malicious.

On the contrary, the key novelty introduced in Bitcoin is a distributed consensus protocol which is Sybil-attack resistant and works in a permissionless environment, where any node, without begin trusted and identified, has the right to participate in the system.

The Bitcoin consensus protocol The first step of the Bitcoin consensus protocol is that all new transactions are broadcast to the Bitcoin peer-to-peer network.

After that, new transactions are collected by nodes called *miners*. A miner is a node that groups new transactions into a block. Any node in Bitcoin can be a miner, no special permission is required.

At this point, the Bitcoin distributed consensus protocol selects the miner which stores the next block in the blockchain. The selection of the miner is done via a cryptographic puzzle. The miner which for first solves the puzzle has the right to append the block it assembled to the blockchain.

The solution to the cryptographic puzzle is called *Proof of Work* (PoW). The puzzle consists in repeatedly computing the hash of the block until the hash results lower than a certain target value. The only way to solve the puzzle is via repetitions: a nonce present in the block is changed until the produced hash is lower than the target.

The probability for a miner to find a PoW is directly proportional to the fraction of the network computational power the miner owns. If a miner with its hardware owns 1% of the total computational power of the Bitcoin network, that miner has the probability to produce one block every hundred.

The PoW is difficult to compute: a new block is produced on average every ten minutes. Periodically, the difficulty is automatically adjusted, to keep the block latency fixed even if nodes computational power increases. The rationale of the ten-minutes latency is to ensure that all nodes in the network have enough time to know the new block and to add it to their own copy of the blockchain, thus keeping all the blockchain replicas consistent.

A miner which finds a PoW and stores a block in the blockchain receives bitcoins as a reward. The rationale is to incentivize miners to produce valid blocks with valid transactions. In fact, an invalid block would be rejected by the network, it would not be added to the blockchain and the miner would not be rewarded. In addition, a misbehaving miner would not only lose the reward, but would also incur in economic loss, as computing the PoW requires expensive specific hardware and consumes significant amount of electricity.

The reward for the miner of a block is constituted by two parts:

- Transaction fees of the transactions included in the block. A transaction fee is computed as the difference between the total amount of bitcoins of the transaction outputs and the total amount of bitcoins referenced by the transaction inputs.

- Freshly minted bitcoins. Each new block has a special transaction, called *coinbase transaction*, which creates a protocol-defined amount of new bitcoins. The coinbase transaction has no input and the output address can be decided by the miner which assembled the block.

To recap, these are the steps of the Bitcoin protocol, as summarized in [1]:

1. New transactions are broadcast to the Bitcoin peer-to-peer network.
2. Each miner collects new transactions into a block.
3. Each miner works on finding a Proof of Work for its block.
4. When a miner finds a PoW, it broadcasts the block to the Bitcoin network.
5. Nodes accept the block only if all transactions in it are valid.
6. Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

By means of this protocol, Bitcoin works in a permissionless decentralized environment. Any node that has sufficient computational and bandwidth capacities can verify new transactions and blocks and insert them in the blockchain, without having specific permissions or being trusted.

Forks If two miners solves the PoW in the same moment, a fork in the blockchain occurs, as Figure 2.3a shows: the two new blocks, B1 and B2, both have a reference to the same previous block.

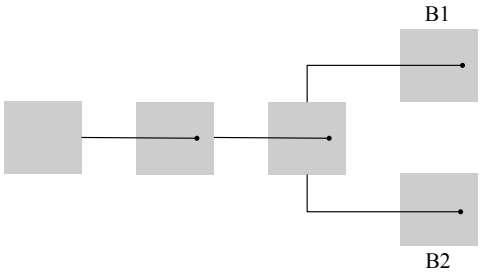
Forks are not permitted in Bitcoin as they break consensus and may cause double spending. It may happen, in fact, that in the two blocks of the fork there are two transactions spending the same bitcoins, and it is not clear which one of the two is valid.

The solution adopted in Bitcoin is that the longest branch is valid. As Figure 2.3b shows, after a fork, the miner producing the next block (B3) inserts the hash of one of the two competing blocks (B2) in that block. In this way, block B1 becomes part of the shortest branch, and is discarded. Transactions in the discarded block are not considered confirmed.

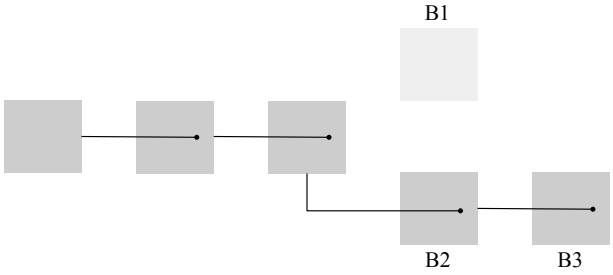
Six confirmations To be sure that a transaction is stored in the longest valid chain and not discarded, it is recommendable to wait for six confirmations.

A transaction is said to have one confirmation when the block containing it is added to the blockchain. Each other block extending the chain which the transaction block belongs to adds one more confirmation.

A transaction with six confirmations will very likely remain in the valid blockchain, as a fork made of more than six blocks is very improbable to happen.



(a) Fork occurrence.



(b) Fork resolution.

Figure 2.3: A fork in the blockchain.

51% attack In Bitcoin a 51% attack refers to the situation in which a single entity owns more than 50% of the network computational power.

In this case, the attacker cannot create invalid blocks or invalid transactions, as they would be rejected by the other nodes. It cannot steal others' bitcoins, as it does not know other nodes private keys.

However, the attacker has the monopoly on the production of blocks. For this reason, it could cause forks which undermine consensus. It could succeed in creating a branch of blocks which overcome the current valid branch. Transactions in the current branch, which were thought confirmed, are instead discarded, thus destabilizing the payment system and causing possible double spending.

Decentralization The key novelty and fundamental feature of the blockchain is decentralization, that is achieved via a novel distributed consensus protocol which combines Proof of Work and economic incentives. Ideally, such protocol aims to ensure that the Bitcoin payment system works without any central trusted entity, in a permissionless way: in principle any node, without being identified and requiring special permissions, can actively participate to the system.

However, in practice, although the blockchain limits centralization, Bitcoin is not purely decentralized, as Gervais *et al.* show in [11]. In particular, the authors

prove that the top-three (centrally managed) group of miners control more than 50% of the Bitcoin computational power. In addition, they demonstrate that updates and incident resolution are controlled by a small number of administrators which have specific function and influence in the Bitcoin community.

Attacks to Bitcoin It has been systematically proved that Bitcoin may be vulnerable to certain attacks.

One of these attacks is called *selfish mining* [8]. The authors show via simulations that a group of miners, even without owning the majority of Bitcoin computational power, can successfully cause forks. The attacking miners can succeed in incentivizing rational miners to mine blocks in their forked branch.

Another attack is called *eclipse attack* [17]. In such attack, an adversary that controls a sufficient number of IP addresses monopolizes connections to a Bitcoin node. In this way, the adversary filters the blockchain view of the victim node, which thus can be exploited for performing attacks on Bitcoin, such as double-spending and forks.

In [13], the authors show that an attacker could delay delivery of blocks or transactions to other nodes in the Bitcoin peer-to-peer network. This could bring to: more advantages in selfish mining, if the attacker is able to avoid delivering of blocks from honest miners to a portion of the network; denial of service, because, if the attacker controls several nodes, it can prevent dissemination of information.

Anonymity is yet another open issue in Bitcoin. Some works prove that it is possible to link a Bitcoin address (a public key) to the IP address of its owner through analysis of network traffic [22], and it is possible to discover that different Bitcoin addresses are actually owned by the same user [18].

2.2 The Issue of Scalability

In Bitcoin the transaction throughput is capped to a maximum of around seven transactions per seconds [27]. Such transaction throughput is far lower than the thousands of transactions per second which world-wide payment systems are supposed to support [39]. This prevents Bitcoin from scaling.

Two parameters caps the transaction throughput of the Bitcoin blockchain. First, the block size: the maximum allowed block size, specified in the Bitcoin protocol, is less than 4 MB. Secondly, the block latency: because of the mechanism of the distributed consensus protocol, a block is found on average every ten minutes

The reason of imposing a block size limit is to keep the decentralization degree high. In [5] the degree of decentralization of Bitcoin is measured as the number of functioning nodes in the network. Functioning nodes are nodes that are able to receive new blocks and transactions and validate them. The higher the number of functioning nodes, the higher the degree of decentralization, because the tasks of

validating blocks and transactions are charged to several different nodes, instead of a few centralized nodes.

A functioning node must have specific storage, computational and bandwidth requirements. Storage requirements are necessary to save the entire blockchain and the set of unspent transaction outputs. These two databases are needed to validate the new transactions, checking whether a party is not attempting to spend more funds than it owns. Bandwidth capacities are necessary to receive and transmit transactions and blocks. Computational capacities are required to perform cryptographic verification on new transactions and blocks.

The block size influences the amount of storage, bandwidth and computational resources needed to be a functioning node. The higher the block size, the faster the growth of the blockchain, which thus requires more storage capacity. Moreover, the higher the block size, the higher the bandwidth necessary to receive and transmit new blocks on the network.

For this reason, an higher block size produces a corresponding lower decentralization degree of the network, as more nodes would not have sufficient resources to actively participate in the network. They wouldn't have sufficient storage capacity to store the blockchain and sufficient bandwidth to receive new blocks and keep updated. In addition, an higher block size would disadvantage more nodes in the mining competition, since miners with higher bandwidth would more quickly download a new block (thus beginning earlier to mine on a next block), and would faster transmit their mined block to the rest of the network.

There exist at least four possible solutions to the issue of scalability: re-parametrization of the block parameters, alternative consensus protocols, sharding and payment channel networks. They are briefly discussed in the following.

Re-parametrization A possible solution that addresses scalability is to change the parameters capping the throughput: reducing the block latency and increasing the block size.

However, the authors of [5] prove that, to ensure that 90% of the nodes are able to be functioning nodes, block size should not be greater than 4 MB and block latency should not be lower than 12 seconds. This entails a maximum transaction throughput of 27 transactions per seconds. The authors of [12] go further and show via simulations that re-parametrization can increase the Bitcoin transaction throughput up to maximum 60 transactions per second: if higher, the system security can be compromised.

For these reasons, if an high decentralization degree is an essential Bitcoin requirement, re-parametrization of block size and latency could not be the most effective solution to the scalability issue. In fact, the throughput resulting from the re-parametrization is still incomparably lower than the throughput of world-wide payment systems.

Alternative consensus protocols Another category of solutions to the issue of scalability is represented by alternative consensus protocols, which improve or replace the Proof of Work.

For example, Bitcoin-NG [9] constitutes an improvement of the Bitcoin protocol. While keeping the same decentralization degree of Bitcoin, Bitcoin-NG reduces the Bitcoin inter-block latency and increases the transaction throughput, up to the maximum allowed by network and node processing limits.

Proof of Stake (PoS) [14, 21], instead, is an alternative approach to Proof of Work. In PoS, the right of a node to add a block to the blockchain is proportional to the amount of coins owned by the node: it is not necessary to solve a cryptographic puzzle. Therefore, since the computational expense of the PoW is eliminated, PoS allows the increase of transaction throughput. However, the security properties of Proof of Stake approaches constitute an open research field.

Sharding In the field of databases, sharding is a well-known technique that consists in distributing portions of a database across different nodes. Sharding the blockchain [25, 10] could improve scalability, since nodes do not need to process and store the entire blockchain, but only portions of it. Whether sharding approaches preserves blockchain decentralization is an open research question.

Payment channel networks A radically different approach addressing the scalability is constituted by payment channel networks. The goal of such networks is to move as many payments as possible off the blockchain. In fact, throughput of off-chain payments is not capped by the block parameters, as those payments do not need to be stored on the blockchain.

Payment channel networks seem to constitute the most promising solution to the issue of scalability, as they probably do not significantly compromise blockchain security and decentralization. They could enable payments that, with respect to transactions registered on the blockchain, are: cheaper, as lower fees are required than the fees for on-blockchain transactions; faster, as they do not need to be registered on the blockchain; and more privacy preserving, as they are not visible in the public blockchain. Payment channel networks are discussed in detail in Section 2.3.

2.3 Payment Channel Networks

This Section provides the background on payment channel networks and especially the Lightning Network.

Payment channel networks are networks of payment channels where it is possible to route off-chain payments, which are not subject to the blockchain throughput limit.

A *payment channel* is a two-party bidirectional channel whereby two parties can exchange off-chain payments. In the rest of this Section, to explain the operation of a payment channel, the following example of a payment channel between Alice and Berto is used. The first step taken by Alice and Berto to open the payment channel is to fund the channel with some of their bitcoins, for example Alice allocates 0.5 BTC in the channel and Berto allocates 0.5 BTC. In this case, the payment channel has a total *capacity* of 1 BTC, the Alice's *balance* in the channel is 0.5 BTC and Berto's is 0.5 BTC.

Once the channel is funded, when Alice wants to pay Berto 0.1 BTC, instead of issuing a transaction to the blockchain, they update their balances to reflect the transfer of bitcoins. Therefore, after a payment of 0.1 BTC from Alice to Berto, Alice's balance is updated to 0.4 BTC and Berto's balance to 0.6 BTC.

Payment channels are two-party. To avoid that each pair of nodes has to open a payment channel to exchange off-chain payments, payment channels are connected together, thus creating a *payment channel network*. Two parties, albeit not connected by a direct payment channel, can exchange off-chain payments through multiple payment channels which link payment sender and payment recipient. In the rest of this Section, the following example of a payment network is presented: Alice has a channel with Berto, Berto with Carola and Carola with Davide. Even if Alice and Davide are not connected by a payment channel, Alice can send an off-chain payment to Davide, which is routed across multiple intermediary channels: the channel between her and Berto, the channel between Berto and Carola and the channel between Carola and Davide.

The Lightning Network is the mainstream implementation of a payment channel network. The remainder of this Section is organized as follows. First, the Lightning Network protocol details are explained (Section 2.3.1). Finally, a brief literature on payment channel networks is presented (Section 2.3.2).

2.3.1 The Lightning Network

The Lightning Network is the most developed and used payment channel network. It is built on top of the Bitcoin blockchain. The Lightning Network protocol specifies how to open and manage payment channels and how to route off-chain payments in a network of payment channels. In the following, such specifications are described.

The LN Payment Channel

In the following the protocol specifications for managing a payment channel in Lightning Network are presented.

Channel opening To open a payment channel in Lightning Network, a *funding transaction* is created by Alice and Berto. The funding transaction serves for funding the channel. As inputs, it has the funds allocated in the channel by Alice and Berto: 0.5 bitcoin owned by Alice, which constitutes the initial Alice's balance in the channel; 0.5 bitcoin owned by Berto, which constitutes the initial Berto's balance in the channel. The output amount of the funding transaction is 1 BTC, which constitutes the total channel capacity[2]. The output script is a 2-of-2 multisignature: this output can be spent only by a transaction that contains the signatures of both Alice and Berto.

The funding transaction is sent to the blockchain and once confirmed, the channel is considered open.

Alice and Berto also create a *commitment transaction*. The commitment transaction returns the balances of the channel to the respective owners. The commitment transaction input is signed by both Alice and Berto and spends from the funding transaction. The transaction outputs are two: one output returns 0.5 BTC to Alice and the other returns 0.5 BTC to Berto.

It is important to highlight that the commitment transaction is not sent to the blockchain (unless a party wants to close the channel).

Payment execution To execute a payment in the channel, Alice and Berto create off-chain a new commitment transaction with the balances updated. When Alice wants to pay Berto 0.1 BTC, they create a new commitment transaction which returns 0.4 BTC to Alice and 0.6 BTC to Berto.

Such operation is done totally off-chain, without the need for interacting with the blockchain. It is repeated for each payment exchanged between Alice and Berto, in both directions. Unlike on-chain transactions, throughput of such off-chain payments is not bounded (except by the network connection speed of Alice and Berto).

Channel closing When the channel parties want to close the channel, they broadcast the most recent commitment transaction to the blockchain, which returns the balances to the respective owners. Once the commitment is confirmed, the channel is closed.

This can be done also in case a party becomes unresponsive/uncooperative. The other party can still broadcast the most recent commitment to the blockchain and recover its own funds.

[2]In practice it is slightly less, as a certain amount is devoted for the fee necessary to store the transaction on the blockchain

Punishment After Alice pays Berto 0.1 BTC and they create a new commitment with the updated balances, Alice could be incentivized to cheat Berto by sending to the blockchain the previous commitment. Such commitment, in fact, would return her more bitcoins than the new commitment.

For this reason, the Lightning Network protocol allows to punish a cheating party. It allows to revoke the old commitment when a new commitment is created and to punish a party if it broadcasts a revoked commitment. If a cheating party sends a revoked commitment to the blockchain, the channel counterparty can take all the funds in the channel, including the funds of the cheating party, which therefore would lose all its funds.

Punishment guarantees trustlessness: a party does not need to trust its channels counterparty. Even if Alice is not trustworthy and cheats, Berto can recover his funds.

Punishment, however, is not automatic. Berto has to actively monitor the blockchain to check whether a revoked commitment has been broadcast by Alice. If Berto does not notice within a certain timeout that a revoked commitment is present in the blockchain, Alice will succeed in cheating Berto and in taking funds from the revoked commitment.

HTLC Payment Channel Network

The Lightning Network protocol specifies how to route payments across multiple payment channels, thus enabling a payment channel network for off-chain payments. In the following such specifications are described in detail.

HTLC In Lightning Network routing of a payment across multiple channels is done via a specific contract called *Hashed Timelock Contract* (HTLC). The HTLC ensures trustlessness: a party involved in a payment route is guaranteed not to lose money, even in case the other parties in the route are not trustworthy and misbehave.

The HTLC implements off-chain conditional payments in a payment channel. When Alice establishes an HTLC with Berto of value 0.1 BTC, it means that Alice will pay Berto 0.1 BTC if Berto shows a certain value R (called *preimage*). Otherwise, if Berto does not show R within a certain timeout, the payment does not take place.

The HTLC is coded as an output script in the commitment transaction of a payment channel. As already explained, the commitment transaction in the channel of Alice and Berto where Alice's balance is 0.5 BTC and Berto's balance is 0.5 BTC has two outputs: one output that gives 0.5 BTC to Alice; one output that gives 0.5 BTC to Berto.

When an HTLC of 0.1 BTC is established between Alice and Berto, a new commitment transaction is created, made of three outputs: one output which gives

0.4 BTC to Alice; one output which gives 0.5 BTC to Berto; the HTLC output, which contains 0.1 BTC. The HTLC output locking script is constituted by two parts:

- One part contains the hash of R. This ensures that Berto has to know R to correctly compute its hash and take the funds of the HTLC.
- One part contains a *timelock*. A a timelock in Bitcoin implements the timeout, ensuring that after a certain date from the establishment of the HTLC (say, one day), Alice can take the funds of the HTLC[3].

With this construction, if Berto shows R within one day, the HTLC is fulfilled, so the payment takes place, and the value of the HTLC is transferred to Berto's balance. A new commitment transaction is created, where the HTLC output is deleted, an output gives 0.4 BTC to Alice and the other output gives 0.6 BTC to Berto.

On the contrary, if Berto does not show R within one day, the HTLC is failed, so the payment does not take place and the value of the HTLC is transferred to Alice's balance. A new commitment transaction is created where the HTLC output is deleted, an output gives 0.5 BTC to Alice and the other output 0.5 BTC to Berto.

Also payments via HTLC are performed totally off-chain, without any interaction with the blockchain. When establishing, failing or fulfilling an HTLC, each new commitment transaction is created off-chain.

In addition, also payments via HTLC are protected against cheating parties. If a party broadcasts to the blockchain a revoked commitment transaction, the counterparty can take all the channel funds, including the ones of the HTLC.

Multihop payment via HTLCs Figure 2.4 shows a payment of 0.1 BTC routed across multiple channels from Alice to Davide. To do that, an HTLC is established in each channel traversed by the payment. All these HTLCs require the same preimage R to be fulfilled.

First, Davide generates R and gives Alice the hash of R. Secondly, an HTLC with the hash of R is established in all the involved channels. In particular, the following HTLCs are established:

- an HTLC of 0.1 BTC in the channel of Alice and Berto with timelock of, say, 3 days;
- an HTLC of 0.1 BTC in the channel of Berto and Carola with timelock of 2 days;

[3]In practice, the timelock is expressed as number of blocks: after a certain number of blocks a transaction output can be spent.

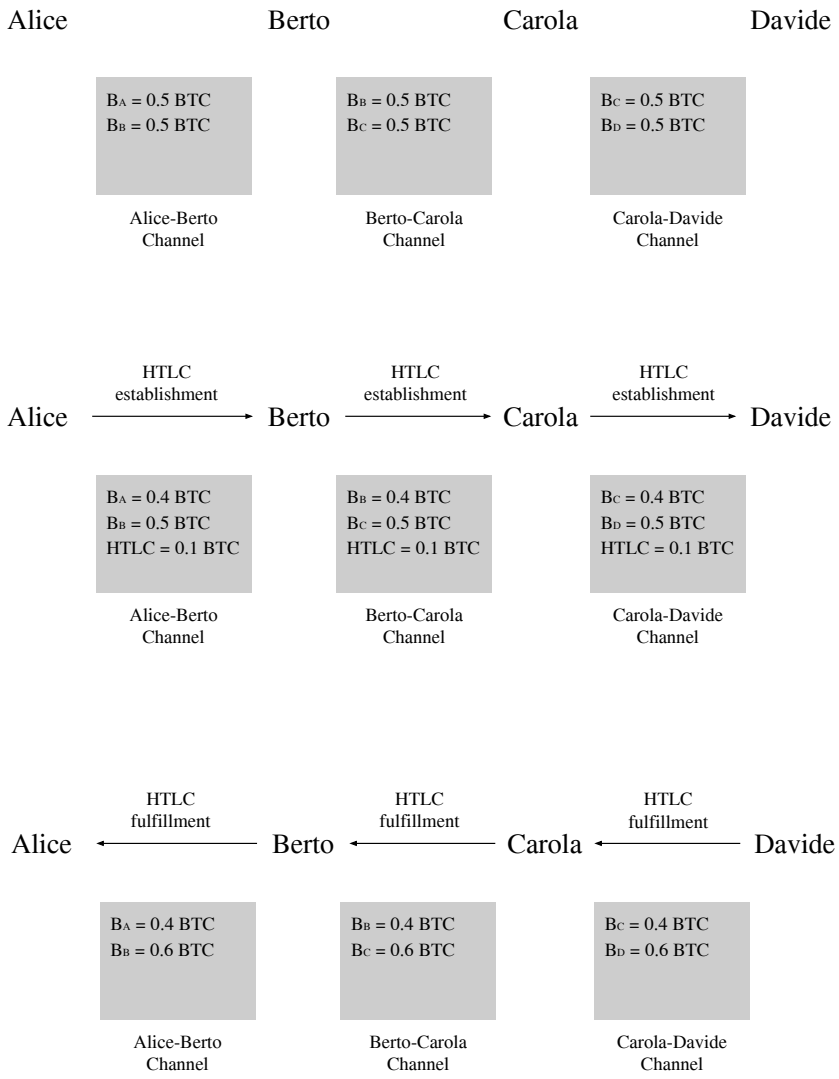


Figure 2.4: Multihop payment via HTLCs.

- an HTLC of 0.1 BTC in the channel of Carola and Davide with timelock of 1 day.

Finally, HTLCs from Davide to Alice are fulfilled. Davide shows R to Carola within 1 day and gets 0.1 BTC from the HTLC with Carola. Carola shows R to Berto within 2 days and gets 0.1 BTC from the HTLC with Berto. Berto shows R to Alice within 3 days and gets 0.1 BTC from the HTLC with Alice. At the end,

0.1 BTC has been transferred from Alice to Berto.

It is important to notice that, from Davide to Alice, HTLCs have increasing timelocks. This ensures that each party has enough time to know R and claim the funds: Davide shows R to Carola in one day, Carola pays 0.1 BTC to Davide; after that, Carola still has one day to claim 0.1 BTC from Berto, as the timelock with Berto is set to two days.

In the case R is never revealed, HTLCs must be failed and, after the timelock expiration, funds of the HTLC return to the payer in each channel. If Davide does not reveal R, after one day, when the timelock expires, Carola takes back the funds of the HTLC. This can be done cooperatively with Davide, by creating a new commitment transaction off-chain. Instead, if Davide is not cooperative, Carola can send the most recent commitment to the blockchain and closes the channel. Then Carola propagates back the failure, so that the HTLCs in all the other involved channels are failed. It is important to notice that, if R is not revealed, funds in the HTLC stay locked up to the timelock expiration.

The last detail that has still to be mentioned is fees for forwarding off-chain payments in the payment network. When Alice wants to transfer 0.1 BTC to Davide, in practice she adds a small amount of bitcoins, which serve as a fee for paying all the intermediary nodes (Berto and Carola) that forward the payment in their channels.

2.3.2 Brief Literature

Although the general idea of a payment channel is as old as Bitcoin and was introduced by Bitcoin inventor Satoshi Nakamoto [15], the first working code for unidirectional micropayment channels dates back to June 2013 [16]. In 2015, Decker *et al.* [6] introduced bidirectional channels and proposed a payment channel network, which enables off-chain payments even between nodes not directly connected by a payment channel. In the same year, the Lightning Network white paper was published by Joseph Poon and Thaddeus Dryja [33]. With respect to Decker's approach, the key novelty of the LN is the possibility of punishing a misbehaving party in a payment channel. Nowadays, the Lightning Network is the mainstream payment channel network.

Although the most developed, the Lightning Network is not the only approach for creating a payment channel network in the literature. Raiden [35] uses smart contracts to implement the same fundamental concepts of the LN on top of the Ethereum blockchain. Sprites [28] is an attempt to improve both the LN and Raiden, as it aims to minimize the time during which funds are locked while in transfer among multiple channels.

The Lightning Network leverages source routing: when routing a payment, the path of channels connecting the payment sender and the payment receiver is found by the payment sender, basing exclusively on its view of the network. There exist

alternatives to this routing approach in the literature. Flare [34] proposes the usage of beacon nodes randomly scattered in the network to increase a node knowledge of the network. In SilentWhispers [26] landmark routing is used: a path between a sender and a receiver is calculated using an intermediary node called landmark, where landmarks are dedicated nodes.

Chapter 3

CLoTH: a Simulator for HTLC Payment Networks

This Chapter introduces CLoTH, a simulator for HTLC payment networks developed in this work. It is organized as follows. Section 3.1 explains `lnd`, the implementation of the Lightning Network taken as reference for developing CLoTH. Section 3.2 illustrates workflow and implementation details of CLoTH. Section 3.3 presents the formal model of the simulator. Section 3.4 discusses the simulator performance in terms of execution time. Section 3.5 explains the assumptions for the simulations performed by CLoTH. Finally, Section 3.6 illustrates the related work on simulations on payment channel networks.

3.1 Analysis of `lnd`

For implementation level details of HTLC mechanisms, `lnd` was taken as reference, which is the implementation of the Lightning Network in Golang language. `lnd`, as the other LN implementations (e.g., `c-lightning`[1] and `eclair`[2]), fully conforms to the so-called *Basis of Lightning Technology* (BOLT) [24], the Lightning Network specifications. The reason of choosing `lnd` is that it is the most documented of the LN implementations in terms of comments to the code.

Figure 3.1 shows the multi-hop payment call graph resulting from the analysis conducted on the release `lnd-v0.5-beta`[3]. The call graph represents the main functions called when a payment flows from a sender to a receiver through an intermediate hop. These functions implement the routing of a payment through HTLCs. (see Section 2.3.1).

[1] https://github.com/ElementsProject/lightning

[2] https://github.com/ACINQ/eclair

[3] https://github.com/lightningnetwork/lnd/releases/tag/v0.5-beta

The figure describes three nodes, as they call different functions with different behaviors: the payment sender $\mathcal{S}$, the payment receiver $\mathcal{R}$ and a generic intermediate hop $\mathcal{H}$. This case can be easily generalized to n-intermediaries since the behaviour of each intermediate hop is always the same.

HTLC messages (i.e., `HTLCAdd`, `HTLCFulfill` and `HTLCFail`) that flow through nodes are represented by arrows. `HTLCAdd` is sent from $\mathcal{S}$ to $\mathcal{R}$ through $\mathcal{H}$ for the establishment of HTLCs among the nodes. `HTLCFulfill` is sent from $\mathcal{R}$ to $\mathcal{S}$ through $\mathcal{H}$, for the fulfillment of the HTLCs. `HTLCFail` is sent in case of failures, for failing the HTLCs.

The function `handleUpstreamMsg` in the gray box represents the first function called by a node when an HTLC message is received. The gray line represents the `commitSig` procedure that establishes an HTLC: the creation of a new commitment transaction containing the HTLC between the two involved parties.

In Appendix A, details of the `lnd` code and of the functions of the call graph are provided, while what follows is an high-level description of the flow of a payment in `lnd`.

Payment routing and sending `lnd` relies on source routing, so the route for a payment must be found by the payment sender. Therefore, when sending a payment, $\mathcal{S}$ first searches for a route connecting it to the payment receiver $\mathcal{R}$.

$\mathcal{S}$ uses its view of the network when searching for the route. The view is built with the channel announcements received: each time a new channel is open, the channel is announced to the network via a gossip protocol. A *channel announcement* contains all the information on the channel such as the channel capacity and the two sets of channel policies, one for each channel direction. A *channel policy* represents the set of policies applied by a channel node to the payments that the node forwards in that channel. Two of these policies are the fees and the timelock applied to the HTLCs that traverse the channel.

Using all these data, $\mathcal{S}$ runs a modified version of Dijkstra's algorithm to find the shortest route to $\mathcal{D}$. The distance metric adopted in that modified version is a combination of fee and timelock, which aims at preferring channels that apply lower fees and lower timelocks.

During Dijkstra's algorithm execution, the node excludes a channel if it has not enough capacity to forward the payment or if the payment amount is lower than the minimum amount allowed by the channel policy. It excludes also a set of nodes and channels that were blacklisted in previous payment executions (for the reasons of blacklisting, see the following paragraph on payment re-attempt).

It is possible that the payment receiver generated an invoice for the payment and included some routing hints in the invoice. In this case, Dijkstra's algorithm uses also these routing hints when searching for a route.

If a route capable of transferring the payment is not found, the payment fails. Otherwise, $\mathcal{S}$ crafts an `HTLCAdd` packet which is forwarded hop-by-hop up to $\mathcal{R}$

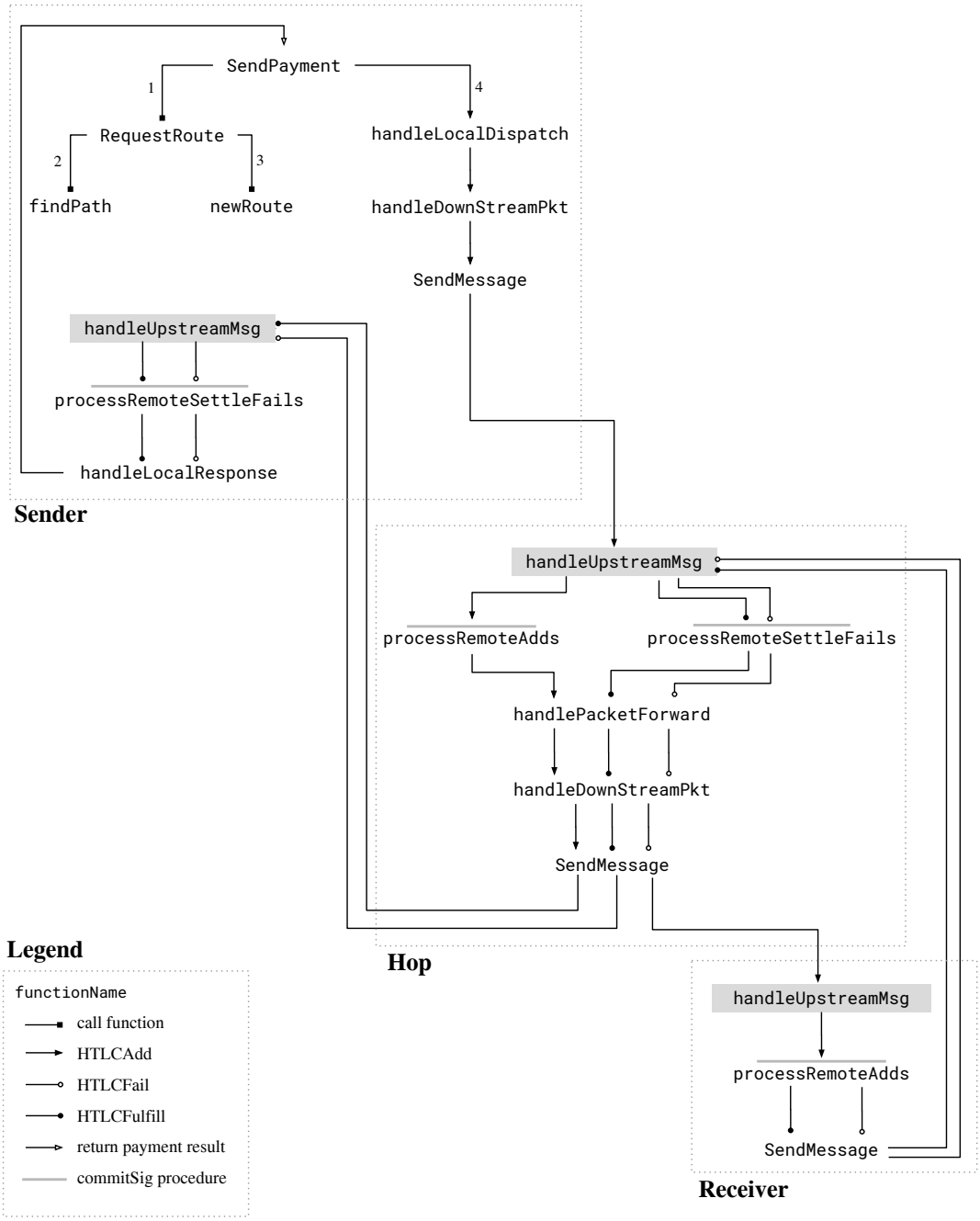


Figure 3.1: Multi-hop payment call graph of lnd.

to establish the HTLCs in the channels of the route. The sender builds all the HTLCs with the values of timelocks and fees that respect the channel policies of

the channels in the route.

`lnd` uses *Sphinx* to exchange HTLC messages. Sphinx is an onion-routing protocol which ensures privacy of payments. In particular, it ensures that each intermediary hop in the route only knows the previous and next hop of the route: it does not know nor its position in the route, neither the length of the route. The drawback of Sphinx is that it limits the maximum length of a route to twenty hops.

Finally, it is worth noticing that in the `lnd` release examined, the maximum amount of payment that can be sent over the Lightning Network is 4.29 millions of satoshis[4].

Payment forwarding The generic intermediate hop $\mathcal{H}$ in the route is in charge of forwarding the payment. Before forwarding the payment in the channel to the next node, it checks for possible errors. Among the others, it checks:

- whether it has enough balance in the channel to forward the payment;
- whether the HTLC satisfies its channel policy.

If the conditions are not met, it propagates an `HTLCFail` message to the previous node in the route, until it arrives to the payment sender. This message fails the HTLCs established for the payment. Otherwise, it forwards the `HTLCAdd` message to the next node in the route.

Payment reception Upon receiving an `HTLCAdd` message, the payment receiver $\mathcal{R}$ checks for possible errors before accepting the payment. Among the others, before establishing the HTLC in its channel, it checks whether the timelock for the HTLC has not already expired.

If there are errors, it propagates back an `HTLCFail` message. Otherwise, it propagates back an `HTLCFulfill` message, which fulfills the HTLCs for the payment and executes the payment.

Payment re-attempt When the payment sender $\mathcal{S}$ receives an `HTLCFail` for its payment, it may re-attempt the payment. It searches for a new route, possibly excluding from the search nodes and channels which caused the payment failure and which therefore are blacklisted. If no new route is found, the payment is considered definitively failed.

Possible errors during a payment attempt are the following:

- An intermediate node claims it did not receive enough fee. In this case $\mathcal{S}$ update its information on the fee policy of the node. If this same error is received again by the same node, the node is blacklisted.

[4] 1 *satoshi* corresponds to 10^{-8} bitcoins.

- An intermediate node claims that the HTLC timelock for its channel does not respect its channel policy. In this case, $\mathcal{S}$ blacklists the node.
- There is not enough balance in a channel to forward the payment. $\mathcal{S}$ blacklists the channel.
- A node in the route is unknown or offline. In this case, $\mathcal{S}$ blacklists the channel connecting to that node.

$\mathcal{S}$ removes nodes and channels from the blacklist after a certain expiration time. For nodes, this expiration time is five minutes; for channels, it is five seconds.

In any case, the payment is not re-attempted if a timeout of 60 seconds has expired from the first payment attempt.

3.1.1 Update of CLoTH

For the simulations in Chapters 4 and 5 an earlier version of the simulator was used, which took as reference a previous version of `lnd`[5]. For simulations in Chapter 6, the simulator was updated to reflect a more recent version of `lnd` (`lnd-v0.5-beta`). The following are the main updates implemented in CLoTH:

- A payment fails if a timeout of 60 seconds expires.
- Blacklisted nodes and edges are removed from the blacklist after five minutes and five seconds respectively.
- The minimum HTLC channel policy is introduced.
- Dijkstra's algorithm considers also fees as distance metric (while in the previous version of `lnd` it considered only timelocks).

3.2 The CLoTH Simulator

In this Section, a detailed description of the CLoTH simulator is provided[6]. Here, the HTLC payment network is defined as a payment channel network where off-chain payments are routed using HTLC contracts.

The CLoTH simulator is written in C. It takes as input a definition of HTLC network and payment script to be played during simulation. It simulates payments in the HTLC network by locally running a discrete-event mapping of the `lnd` code.

[5] https://github.com/lightningnetwork/lnd/commit/4d6cd2ee36885d3df4d38fc90fd3058885486d83

[6] The code of the simulator in pre-alpha is available online at https://researchdata.nexacenter.org/payment-network-simulator/cloth-0.0.2.zip

It produces performance measures in the form of payment-related statistics (e.g., the probability of payment failures and the mean payment complete time).

The computation flow of the simulator is constituted by three phases, as Figure 3.2 shows. In the rest of this Section, each phase is explained.

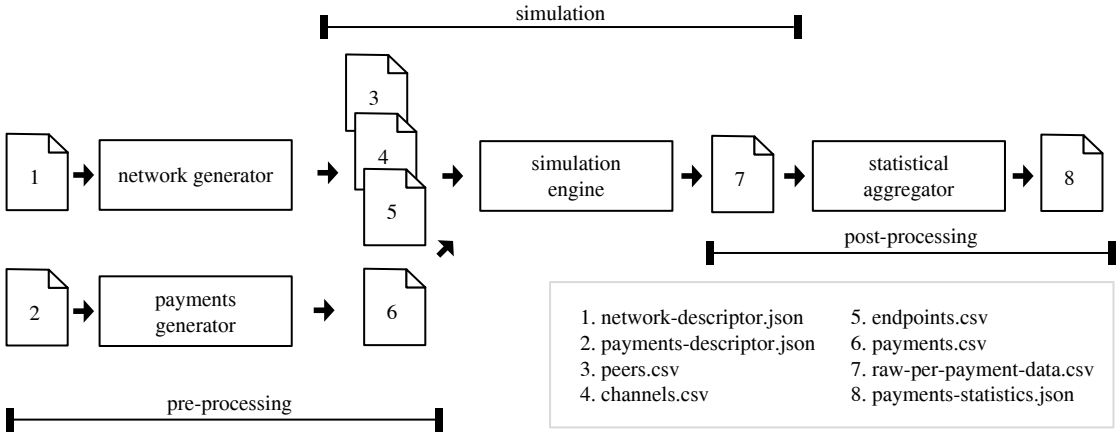


Figure 3.2: CLoTH simulator workflow.

3.2.1 Pre-Processing Phase

The pre-processing phase serves for generating the HTLC payment network and the payments which are executed in the network during the simulation phase.

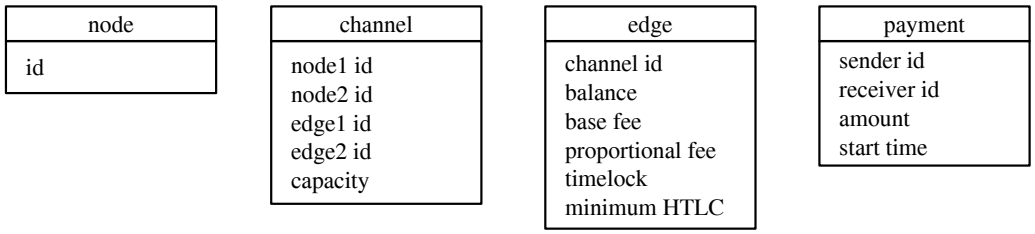


Figure 3.3: CLoTH simulator data structures.

Data structures The payment channel network and the payments are represented in the simulator by the data structures in Figure 3.3. A `channel` connects two `node` (each one represented by an ID) and has a certain capacity. In addition, since a channel is bidirectional, namely, payments can traverse it both from `node1` to `node2` and from `node2` to `node1`, it contains two edges, each one representing a direction of the channel. An `edge` contains: the ID of the `channel` the edge

belongs to; the available balance in the direction represented by the edge; base and proportional fee, which constitute the fee required for forwarding a payment in the direction of the edge (proportional fee is the part of the fee which depends on the payment amount, while base fee is the constant fee applied regardless of the payment amount); the timelock set in the HTLCs established in the direction of the edge; and the minimum value allowed for payments forwarded in the direction of the edge. A `payment` is described by a sender, a receiver, the payment amount and the payment start time, that is the instant in which the payment occurs in the simulator.

Input modes There are two possible input modes for populating the simulator data structures:

1. by directly providing a complete specification of each attribute of the data structures (files 3, 4, 5 and 6 in Figure 3.2);

2. by providing few input parameters that statistically define an HTLC network and a script of payments to be simulated (files 1 and 2 in Figure 3.2). In this case, the network and payment generator engines of the simulator - using random variables - generate an instance of HTLC network and an instance of payment script that match the input description.

Although verbose, the first input specification allows the most detailed definition of the simulation scenario. This is useful to simulate payments on an existing payment channel network, fetching the nodes and channels data of the network and providing them as input to the simulator. It is also useful to modify specific attributes of the network or payments generated by the respective generators, e.g., by removing a certain node or increasing a certain channel capacity. The second input mode, instead, allows a concise description of a simulation from a statistical point of view, using just a few input parameters.

Table 3.1: CLoTH simulator input parameters.

Type	Symbol	Definition
Network	N_n	Number of nodes
	N_{ch}	Average number of channels per node
	σ_t	Tuner of network topology
	$P_{\bar{c}_b}$	Uncooperative nodes probability before HTLC establishment
	$P_{\bar{c}_a}$	Uncooperative nodes probability after HTLC establishment
	C_{ch}	Average channel capacity
	G	Gini index of channel capacity
Payments	r_π	Payments per second (off-chain)
	N_π	Number of payments
	σ_a	Tuner of payment amounts
	F_{sr}	Fraction of same-recipient payments

Input parameters Table 3.1 shows the input parameters of the simulator with their symbols . Here follows an explanation of the parameters:

- N_n is the total number of nodes in the payment network.
- N_{ch} specifies the average number of channels per node. For each node, its channels counterparties are chosen using a uniform random distribution, except one, which is selected with a Gaussian distribution (see below the explanation of σ_t).
- σ_t tunes the presence of hubs in the network topology. It is the width of the Gaussian distribution defining the probability of connection among nodes. For each node, this Gaussian probability distribution is used for choosing the counterparty of one of the channels of the node. Therefore, if the width of this Gaussian is zero, all nodes have one channel open with the same node, which, consequently, will be a hub. On the other hand, if the Gaussian width is infinite, any node has the same probability to be connected to any other, thus producing a totally decentralised network.
- $P_{\bar{c}_b}$ and $P_{\bar{c}_a}$ are the probabilities that a node, when interested in a payment, becomes uncooperative (because it is malicious or faulty). The first is for uncooperativeness before the establishment of an HTLC, the second after the establishment. For the behavior of the protocol in case of uncooperativeness, see Section 3.2.2.
- C_{ch} is the average channel capacity.

- G models the distribution of funds in the channels. The total available funds (given by the average channel capacity multiplied by the total number of channels in the network) are distributed in the channels such as to produce a certain level of Gini index.
- r_π is the average number of payments per second. In particular, the payment inter-arrival time is modeled as a negative exponential random distribution.
- N_π is the total number of payments simulated during a simulation.
- σ_a tunes payment amounts. It is the width of the Gaussian distribution whose tail is used to choose the orders of magnitude of payment amounts. The greater this width is, the higher the payment amounts.
- F_{sr} is the fraction of the total payments directed toward the same recipient. By means of this parameter, it is possible to model the use case of many small payments sent to the same destination node, e.g., to a provider of video-streaming services which is paid for each short segment of video streaming.

If the second input mode is used, the simulator sets also the values of the following attributes:

- Channel balances. Having defined the capacity of a channel, to establish the two edge balances as fractions of the capacity, a gaussian distribution is used, which most probably produce equally balanced channels.
- Base and proportional fees. Base fees are random values uniformly distributed between 1000 and 5000 millisatoshis; proportional fees are random values uniformly distributed between 1 and 10 millisatoshis. These are the typical fee values currently adopted in the Lightning Network.
- Timelock. Timelock of the edges are between 10 and 100 number of blocks. These are the orders of magnitude of timelocks currently used in LN.
- Minimum HTLC. The simulator specifies values of the minimum HTLC policies between 0 and 1000 millisatoshis (low values are more probable than high ones). Also these are the typical minimum HTLC values currently used in LN.
- Network latency. For the inter-node communication the simulator sets a network latency uniformly distributed among 10 and 100 milliseconds, which is a reasonable latency value considering that `lnd` uses onion routing.
- Faulty latency. This is the time waited when trying to contact a node which is faulty/malicious and does not respond. It is set to 3 seconds, that is the typical TCP faulty time.

Multi-thread execution Finally, before the simulation phase, once the payments have been defined, the simulator runs in multi-thread to compute the initial routes for all the payments. The rationale of this design choice is that Dijkstra's algorithm is the most computationally expensive and time consuming task of the simulator, and it is convenient to execute it in parallel.

3.2.2 Simulation Phase

During the simulation phase, the simulator simulates the execution of the input-defined payments in the input-defined HTLC network. In particular, the simulator runs a *discrete-event* simulation.

Discrete-event simulation *Events* are instantaneous time occurrences which change the state of the system. It is discrete because the *simulation time*, that is the time represented in the simulation, advances through discrete steps from one event to the next.

The core engine of the discrete-event simulator is the *event scheduling*. It is modeled as a queue, in which events are inserted and ordered according to their instant of occurrence. When an event is extracted from the queue, the simulation time is advanced to the instant of occurrence of the event and the event is processed. Processing of an event may cause the generation of a new event, which is inserted in the event queue. The simulation ends when a pre-defined simulation end time is reached

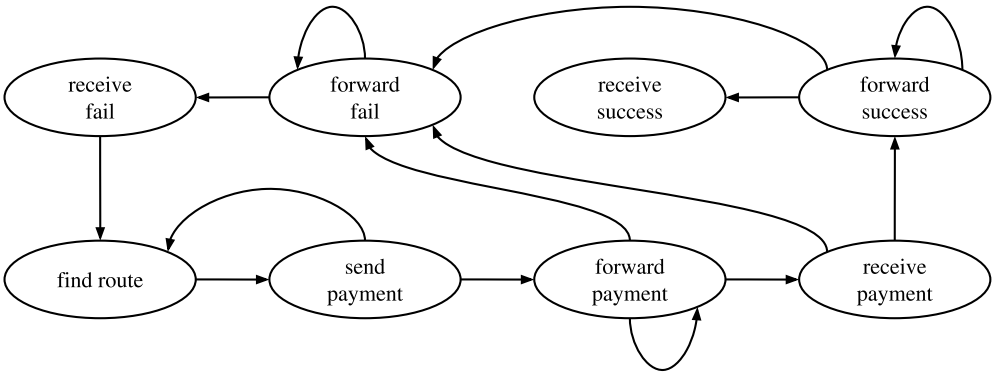


Figure 3.4: CLoTH simulator events state diagram.

In the CLoTH simulator, an event represents a state of a payment. An event is generated each time a payment changes its state according to the state diagram in Figure 3.4, and it is processed by a function of the same name of the event. For example, the event "find route" represents a payment for which a route has to be found. When such event is extracted from the queue, a function of the same name is

called, which searches for a payment route. If the route is found, a "send payment" event is generated for that payment, which represents the state of a payment that has to be sent by the payment sender. And so on, the payment flows as described in Section 3.1.

Table 3.2: Mapping between CLoTH simulator functions and `lnd` functions.

Simulator Function	**Description**	**Simulated Functions**	**Node**	**Message**
`find_route`	Search for a payment route	`SendPayment`, `RequestRoute`, `findPath`, `newRoute`	Sender	-
`send_payment`	Send a payment	`handleLocalDispatch`, `handleDownStreamPkt`, `SendMessage`	Sender	`HTLCAdd`
`forward_payment`	Forward a payment	`handleUpstreamMsg`, `processRemoteAdds`, `handlePacketForward`, `handleDownStreamPkt`, `SendMessage`	Hop	`HTLCAdd`
`receive_payment`	Receive a payment	`handleUpstreamMsg`, `processRemoteAdds`, `SendMessage`	Receiver	`HTLCAdd`
`forward_success`	Forward the successful result of a payment	`handleUpstreamMsg`, `processRemoteSettleFails`, `handlePacketForward`, `handleDownStreamPkt`, `SendMessage`	Hop	`HTLCFulfill`
`forward_fail`	Forward the fail result of a payment	`handleUpstreamMsg`, `processRemoteSettleFails`, `handlePacketForward`, `handleDownStreamPkt`, `SendMessage`	Hop	`HTLCFail`
`receive_success`	Receive the successful result of a payment	`handleUpstreamMsg`, `processRemoteSettleFails`, `handleLocalResponse`	Sender	`HTLCFulfill`
`receive_fail`	Receive the fail result of a payment	`handleUpstreamMsg`, `processRemoteSettleFails`, `handleLocalResponse`	Sender	`HTLCFail`

Functions of the simulator Table 3.2 shows the functions of the simulator that process the events with their description. For each simulator function, the table shows the functions of `lnd` which are simulated by the considered simulator

function. The completeness of the mapping between functions of CLoTH and each function of the `lnd` computation flow ensures the validity of the simulated results.

Table 3.2 makes a clear distinction between specific behaviors of each `lnd` function which depends on two parameters: the type of node that invokes it (i.e., sender, receiver or intermediate hop) and the type of the triggering message (`HTLCAdd`, `HTLCFail` or `HTLCFulfill`). For example, the function `send_payment` of the simulator simulates the functions `handleLocalDispatch`, `handleDownStreamPkt` and `SendMessage` of the `lnd` code, when they are called by the payment sender in the case of `HTLCAdd` message.

Listing 3.1 presents the simplified C code of the simulator functions. Here follows a detailed explanation of each function.

```
void find_route(){
  if(payment->duration > 60000) {
    register_payment_stats();
    return;
    }

  update_blacklist();

  route = dijkstra();
  if(route == NULL){
    register_payment_stats();
    return;
  }

  next_event_time = current_time;
  create_event(send_payment, next_event_time);
}


void send_payment(){

  if(unknown_forward_edge){
    blacklist(forward_edge);
    next_event_time = current_time;
    create_event(find_route, next_event_time);
    return;
  }

  if(forward_edge->balance < payment->amount){
    blacklist(forward_edge);
    next_even_time = current_time;
    create_event(find_route, next_event_time);
    return;
```

```
  }

  forward_edge->balance -= payment->amount;

  next_event_time = current_time + network_latency;
  create_event(forward_payment, next_event_time);
}


void forward_payment(){

  if(uncooperative_before_HTLC){
    blacklist(edge);
    next_event_time = current_time + faulty_latency;
    create_event(receive_fail, next_event_time);
    return;
  }

  if(uncooperative_after_HTLC){
    blacklist(edge);
    next_event_time = current_time + timelock;
    create_event(receive_fail, next_event_time);
    return;
  }

  if(unknown_forward_edge){
    blacklist(forward_edge);
    next_event_time = current_time + network_latency;
    create_event(receive_fail, next_event_time);
    return;
  }

  if(forward_edge->balance < payment->amount){
    blacklist(forward_edge);
    next_even_time = current_time + network;
    create_event(find_route, next_event_time);
    return;
  }

  forward_edge->balance -= payment->amount;

  next_event_time = current_time + network_latency;
  create_event(receive_payment, next_event_time);
}
```

```
void receive_payment(){
  if(uncooperative_before_HTLC){
    blacklist(edge);
    next_event_time = current_time + faulty_latency;
    create_event(forward_fail, next_event_time);
    return;
  }

  if(uncooperative_after_HTLC){
    blacklist(edge);
    next_event_time = current_time + timelock;
    create_event(forward_fail, next_event_time);
    return;
  }

  backward_edge->balance += payment->amount;

  next_event_time = current_time + network_latency;
  create_event(forward_success, next_event_time);
}


void forward_success(){
  if(uncooperative_after_HTLC){
    blacklist(edge);
    next_event_time = current_time + timelock;
    create_event(receive_fail, next_event_time);
    return;
  }

  backward_edge->balance += payment->amount;

  next_event_time = current_time + network_latency;
  create_event(receive_success, next_event_time);
}


void receive_success(){
  register_payment_stats();
}


void forward_fail(){
  forward_edge->balance += payment->amount;

  next_event_time = current_time + network_latency;
```

```
    create_event(receive_fail, next_event_time);
}


void receive_fail(){
    forward_edge->balance += payment->amount;

    next_event_time = current_time;
    create_event(find_route, next_event_time);
}
```

Listing 3.1: The CLoTH Simulator Functions.

Function `find_route()` This function simulates the search for a payment route from the payment sender. It is called either for a new payment or for a payment which previously failed and has to be re-attempted.

First, the function checks whether more than 60 seconds are elapsed since the first payment attempt. If so, the payment is not re-attempted and fails definitely. Therefore, the function `register_payment_stats()` is called to register the statistics for the payment, which will be useful to produce the simulator performance measures (e.g., the payment end time and the reason of the failure - see Section 3.2.3 for the complete list of payment statistics).

Then, the function updates the blacklist of nodes and edges, removing nodes and edges that have been staying in the list for more than 5 minutes and 5 seconds respectively.

At this point, the function calls the modified version of Dijkstra's algorithm to find a payment route, considering fees and timelocks as distance metric and excluding blacklisted nodes and edges. If no route is found, the payment fails.

Finally, the function creates a `send_payment` event.

Function `send_payment()` This function simulates the sending of a payment by the payment sender.

The function first checks whether `forward_edge`, the edge where to forward the payment, is known. If not, the edge is blacklisted and a `find_route` event is created, in order to search for a new route without the unknown edge.

The second check regards the balance of the forward edge. If the balance is lower than the payment amount, the edge is blacklisted and a `find_route` event is created to search for a new route.

It is important to notice that, in each hop of the route, `payment->amount` takes into account also the correct amount of fees to be payed to the hop.

If the previous checks passed, the function decreases the forward edge balance by the payment amount, since an HTLC of amount equal to the payment amount is established in the edge. Finally, a `forward_payment` event is created.

Function `forward_payment()` This function simulates the forwarding of a payment by an intermediate hop in the payment route.

The first two checks of the function are on cooperativeness. In particular, using probability distributions (see $P_{\bar{c}_b}$ and $P_{\bar{c}_a}$ in Section 3.2.1), the function checks whether the hop is uncooperative before or after establishing the HTLC. In the first case, the `receive_fail` event is scheduled after a faulty latency; in the second case, after the timelock, since the HTLC is already established and the timelock must expire before unlocking the funds. In both cases, the function blacklists the edge directed to the current intermediate hop.

As in function `send_payment()`, another check is on the presence of the forward edge. If the forward edge is not known, a `receive_fail` event is created, which informs the payment sender on the payment fail. If there are other intermediate hops in the route between the current hop and the payment sender, the function generates a `forward_fail` event, to propagate the the payment fail to the previous hop.

The last check verifies whether there is enough balance in the forward edge, as done in function `send_payment()`.

After all the checks, the function reduces the balance of the forward edge by the payment amount and generates a `receive_payment` event (or another `forward_payment` event in case there are other intermediate hops before the payment receiver).

Function `receive_payment()` This function simulates the reception of a payment by the payment receiver.

After the checks on the uncooperativeness, the function increases the balance of the backward edge of the receiver channel, which is the edge in the reverse direction of the payment, pointing to the previous hop of the route.

Finally, the function generates a `forward_success` event.

Function `forward_success()` This function simulates the forwarding of a payment success by an intermediate hop of the payment route.

First, the function checks the uncooperativeness. It is important to notice that in this case a node can be uncooperative only after establishing the HTLC, which in fact was already established when forwarding the payment.

If the intermediate hop is not uncooperative, the function reduces the balance of the backward channel in the route. Moreover, the function generates a `receive_success` event (or another `forward_success` event if there are other intermediate hops in the route to which forward the payment result).

Function `receive_success()` This function simulates the reception of a payment success by the payment sender. The function registers the statistics on the payment and does not generate further events, as the payment is complete.

Function `forward_fail()` This function simulates the forwarding of a payment fail by an intermediate hop in the payment route.

It increases the balance of the edge where the payment was previously forwarded, to recover the balance state preceding the payment execution. Finally, it generates a `receive_fail` event (or a `forward_fail` event in case there are other intermediate hops in the route).

Function `receive_fail()` This function simulates the reception of a payment fail by the payment sender.

It increases the balance of the edge where the payment was forwarded. It generates a `find_route` event, in order to re-attempt the payment.

3.2.3 Post-Processing Phase

The post-processing phase of the simulator transforms the raw per-payment output attributes, collected during the simulation phase, into statistically meaningful performance measures.

Output attributes The per-payment output attributes generated by the simulation phase (File 7 in Figure 3.2, `raw-per-payment-data.csv`) are the following:

- the payment end time;
- the payment result: success or failure (and the reason of failure);
- number of times the payment was attempted;
- whether the payment encountered an uncooperative node;
- whether the payment timeout expired;
- route traversed by the payment, if present.

To transform these attributes into performance measures (File 8 in Figure 3.2, `payments-statistics.json`), the *batch means* method [19] is used. Using this method, it is possible to produce performance results that are not influenced by the initial transient state, where the system is not stable, and to compute statistical mean, variance and 95% confidence interval for each measure. The batch means method consists in dividing a simulation run into multiple batches, which are statistically independent among each other. Output measures are zeroed and re-computed at each batch. Each final output measure is the statistical mean of that measure over the batches and comes also with variance and 95% confidence interval.

Simulator performance measures Table 3.3 shows the performance measures produced by the simulator with their symbols. Here follow some clarifications:

- P_{f_r} is the probability that a payment fails due to the absence of a route connecting sender and receiver.
- P_{f_b} is the probability that a payment fails because a channel in the route was unbalanced (namely, its balance was lower than the payment amount) and an alternative route without the unbalanced channel is not found.
- $P_{f_{\bar{c}}}$ is the probability that a payment fails because a node in the route was uncooperative and an alternative route is not found.
- $P_{\bar{t}}$ is the probability that a payment fails because it took more than a timeout of 60 seconds to complete.
- $P_{\bar{k}}$ represents the remaining fraction of payments for which it is impossible to know whether they failed or succeeded, as they ended after the time window of the simulation. For example, this category can encompass payments delayed after a long timelock, as a node was uncooperative after establishing the HTLC for the payment.
- T is the average time for a successful payment to complete.
- N_{att} is the average number of times a successful payment is re-attempted.
- L_r is the average route length traversed by a successful payment.

Table 3.3: CLoTH simulator performance measures.

Symbol	Definition
P_s	Probability of payment success
P_{f_r}	Probability of payment failure for no route
P_{f_b}	Probability of payment failure for unbalancing
$P_{f_{\bar{c}}}$	Probability of payment failure for uncooperative nodes
$P_{\bar{t}}$	Probability of payment failure for timeout expiration
$P_{\bar{k}}$	Probability of unknown payments
T	Payment complete time
N_{att}	Number of payment attempts
L_r	Payment route length

3.3 Formal Model

Having a network with N nodes connected by payment channels and having M payments to be executed on this network, the simulator formal model can be expressed using the following formulas:

$$r_k = \phi(p_k, \mathcal{G}_k, b_k) \tag{3.1}$$

$$\mathcal{G}_{k+1} = \mu(p_k, r_k) \tag{3.2}$$

Function ϕ is the function which finds a route r_k for a payment p_k. It takes as input the following parameters:

- $p_k = (n_{i_k}, n_{j_k}, a_k)$ for $k = 1, 2, ..., M$ is the payment for which a route has to be found: n_{i_k} and n_{j_k} are the sender node and the receiver node of the payment, for $i, j = 1, 2, ..., N$ and $i \neq j$; and a_k is the payment amount.
- $\mathcal{G}_k$ is the graph of nodes n_i for $i = 1, 2, ..., N$ connected by payment channels. The subscript k indicates the state of the graph at the moment in which p_k is processed, with the available nodes and channels and their attributes at that moment.
- b_k is the blacklist of possible nodes and channels excluded when searching for a route for p_k (see Section 3.1).

Given the payment p_k and the route r_k, the function μ executes the payment p_k along the route r_k. μ produces a new state of the graph, namely $\mathcal{G}_{k+1}$, because the payment execution causes some changes in the channels involved in the payment, such as the update of channel balances according to payment amount a_k. A whole simulation is therefore the transition from $\mathcal{G}_0$ to $\mathcal{G}_{|M|+1}$, the initial state of the network and the final one (i.e., the state reached after the execution of the last payment), respectively.

3.4 Performance Analisys

Table 3.4 shows the performance of the CLoTH simulator in terms of execution time. Simulation times refer to experiments run on a machine of model DGX-1, manufactured by Nvidia, located in Rome, Italy. The machine is equipped with 80 CPUs of model Intel® Xeon® CPU E5-2698 v4 @ 2.20 GHz and 512 GB of RAM.

Table 3.4a shows that execution time increases with the increase of the number of nodes of the simulated network. This is due to the fact that time spent by Dijkstra's algorithm, used to find payment routes, grows with the number of nodes. With 1 million nodes, a simulation run requires more than four days.

(a)	
Nodes	**Execution Time (h)**
100,000	6.64
200,000	15.92
500,000	49.25
700,000	64.68
1,000,000	104.15

(b)	
Dijkstra calls	**Execution Time (h)**
58927	4.98
303374	28.3

Table 3.4: CLoTH simulator execution time.

Table 3.4b shows that the execution time also increases with the increase of the number of calls to Dijkstra's algorithm. An execution of Dijkstra's algorithm is required each time a payment is attempted. As more and more payments are re-attempted, calls to Dijkstra's algorithm grow as well as simulation execution time.

Therefore, the execution of Dijkstra's algorithm constitutes a simulator performance bottleneck. However, there are no limits to the values of input parameters. The only effect of setting high values (e.g., an high number of nodes or payments) is a correspondingly long execution time.

3.5 Assumptions

An HTLC payment network relies on a blockchain as a securing mechanism for all its payment channels. The underlying blockchain is therefore a fundamental prerequisite of each HTLC payment network.

CLoTH, however, does not consider blockchain interactions during a simulation execution, since the performance measures produced by the simulator are related to payments, which are completely performed off-chain. Simulations run on a network in which no new channel is opened. This condition is implicitly guaranteed if the whole simulation time is shorter than the time required to fund a new channel with an on-chain Bitcoin transaction. An on-chain Bitcoin transaction is usually considered final after 6 confirmations, statistically 50 minutes after the first confirmation. This implies a relationship between simulation duration and likelihood of simulation results: the shorter the simulation, the more plausible the results. Without loss of generality, the channels that might have been established early enough

before the simulation starting time to become operational during the simulation time window are not considered too. Simulations of around 15 minutes are a good compromise between meaningfulness of the experiment and accuracy of results.

Another assumption is that each node has a perfect knowledge of all other nodes and channels in the network. In reality, as already mentioned, new channels and nodes are announced through a gossip protocol. Therefore, real performance may be slightly worse than the one measured by CLoTH, as nodes may have a slightly imprecise knowledge of the network due the nature of the gossip protocol.

3.6 Related Work

Several works in the literature analyzed payment channel networks through simulations.

In [37], the author proposes and evaluates through simulations a routing protocol for payment channel networks. The authors of the Flare routing protocol [34] runs simulations with 100,000 nodes to study the performance of the protocol. In [32] Piatkivskyi *et al.* developed a Lightning Network simulator called Blyskavka to evaluate the approach of splitting payments when routing them in the LN. Their simulator is a multi-agent discrete-event simulator built for general purpose payment network simulations and it also simulates HTLCs. In [41], Ruozhou Yu *et al.* implemented a simulator to evaluate CoinExpress, a payment routing mechanisms for payment channel networks. Their simulator is a payment channel network simulator based on the network simulator `ns-3`. Stasi *et al.* [7] developed a simulator to evaluate improvements to the LN protocol: a novel fee policy that aims at reducing channel unbalancing and a multipath routing payment scheme. Finally, Reynolds [36] developed ocalm code for basic simulations on LN.

Differently from all the above-mentioned simulators, CLoTH is a precise mapping of the Lightning Network code. In particular, the functions of `lnd` coding the modified version of Dijkstra's algorithm and the HTLC scheme are exactly implemented also in CLoTH. This is the original feature of CLoTH which makes it different from the other simulators and ensures the validity of the results produced by the simulator.

Chapter 4

Simulations on the Lightning Network Mainnet

This Chapter presents a group of simulations conducted on the Ligthning Network mainnet, namely, the network of channels and nodes implementing the LN protocol. In this set of simulations, the input of the simulator was constituted by nodes and channels taken from a recent snapshot[1] of the LN mainnet. The goal of this set of simulations was to discover the cases (if any) in which a payment is more likely to fail than to succeed on the LN mainnet.

The Chapter is organized as follows. First, the design of these simulations is discussed in Section 4.1. Then, the results of these simulations are showed in Section 4.2. Finally, Section 4.3 discusses the main findings of the simulations and answers Research Question 1.

4.1 Simulations Design

In this Section, the design choices of the simulations on the LN mainnet are described and motivated. First, the independent variables of the simulations are presented (Section 4.1.1). Then, the strategy adopted to conduct the simulations is discussed and motivated (Section 4.1.2).

4.1.1 Independent Variables

In general, the simulation independent variables are a proper subset of the simulator input parameters, while the dependent variables are the simulator output measures. Specifically for the simulations on the LN mainnet, the only independent variables were the input parameters of the payments and the two probabilities

[1]Downloaded on June 14th 2018 at 14:32 CEST from https://rompert.com/recksplorer/

defining nodes uncooperativeness: all the other simulator input parameters were the fixed attributes of the mainnet nodes and channels.

Each independent variable was studied in a proper *variation interval*. A variation interval is defined by its extreme values and by its sampling rate (i.e., the grain of its subdivisions, how many intermediate steps are considered). Table 4.1 shows the variation intervals for each independent variable.

Table 4.1: Variation intervals of the independent variables.

Variable	Interval	Unity of Measure
$P_{\bar{c}_b}$	[0 0.01 0.1 1.0 10.0]	%
$P_{\bar{c}_a}$	[0.01]	%
r_π	[10 100 1,000]	payments per second
N_π	[10,000 100,000 1,000,000]	-
σ_a	[1 2 3 4 5]	-
F_{sr}	[0 10 20 40 50]	%

To define the intervals of the independent variables, some initial tuning simulations were run. Here follow the rationales of each variation interval:

- Uncooperative nodes probability before HTLC establishment $P_{\bar{c}_b}$. With regard to the choice of the upper limit, a probability that a node is uncooperative more than one every 10 times is unrealistic.
- Uncooperative node probability after HTLC establishment $P_{\bar{c}_a}$. Payments failing for such uncooperative nodes are not captured by simulations since they end after the simulated time interval (because of the timelock). However, it is realistic to have this value different from zero, as it may happen with a certain small probability that a node experiences a fault and goes offline after establishing an HTLC.
- Average payment rate r_π. A payment rate lower than 10 payments per second is too low for a world-wide payment system as LN aspires to be. The highest value is the average rate supported by mainstream traditional payment systems.
- Number of payments N_π. The values of this interval depend on the average payment per second, and on the fact that, for the assumptions made (see Section 3.5), a simulation lasts around 15 minutes.
- Payment amount tuner σ_a. Table 4.2 shows, for each value of σ_a, the payment amounts generated, in the form of: order of magnitude expressed in satoshis

and ratio of payments whose amount has that order of magnitude. The lowest value is chosen to produce only small payments. The highest value is chosen to produce a certain fraction of high payments, however not higher than 0.1 BTC, as the Lightning Network at least at this stage is not supposed to support large payments.

- Fraction of same-recipient payments F_{sr}. With regard to the upper limit, it is unrealistic to send more than half of the total payments to the same recipient during a single simulations.

Table 4.2: Ratio of payments with a certain order of magnitude for each value of σ_a.

Order of Magnitude (Satoshi)	$\boldsymbol{\sigma_a}$				
	1	**2**	**3**	**4**	**5**
10^0	67.83%	35.26%	26.33%	20.38%	18.00%
10^1	27.61%	29.57%	23.76%	19.85%	17.43%
10^2	4.3%	18.80%	18.40%	17.14%	15.53%
10^3	2.6%	8.81%	13.79%	13.88%	14.14%
10^4	0.0%	3.32%	8.80%	11.48%	11.49%
10^5	0.0%	0.98%	5.08%	8.19%	9.69%
10^6	0.0%	0.20%	2.58%	5.55%	8.19%
10^7	0.0%	0.06%	1.26%	3.53%	5.51%

4.1.2 Simulation Strategy

As already stated, the goal of this set of simulations was to discover non-operative cases. Here, the network is defined as non-operative when a payment is more likely to fail than to succeed, which means that the probability of payment success P_s is below 50%. A branch-and-bound-like strategy was adopted to discover non-operative cases.

To do so, for each independent variable a non-stressing and a stressing value are defined. The *non-stressing value* is a value of the variation interval which is *not* supposed to negatively impact performance. For example, the non-stressing value for the uncooperative nodes probability is zero, as in this case all nodes are cooperative and do not cause payment failures. The *stressing value* is a value of the variation interval which is supposed to negatively impact performance. For example, the stressing value for the probability of uncooperative node is 10%, as it is the highest probability of uncooperative behavior specified in the variation interval.

Table 4.3 shows the full list of stressing and non-stressing values for the independent variables of these simulations.

Table 4.3: Stressing and non-stressing values of independent variables.

Variable	Non-Stressing Value	Stressing Value
r_π	10	100
N_π	10,000	100,000
σ_a	1	5
F_{sr}	0%	50%
$P_{\bar{c}_b}$	0%	10%

The branch-and-bound tree of the simulations performed is depicted in Figure 4.1. Each node represents a simulation, and the name of the node refers to the independent variable stressed in that simulation. The simulation strategy consisted of the following steps:

- First, the non-stressing simulation was run, where all variables were set to their respective non-stressing values.
- Secondly, one independent variable at a time was stressed, thus forming a branch for each simulation with one stressed independent variable.
- If the simulation with the stressed variable produced a non-operative case, no further simulation were performed in the branch. Otherwise, that branch was examined, stressing additional variables.

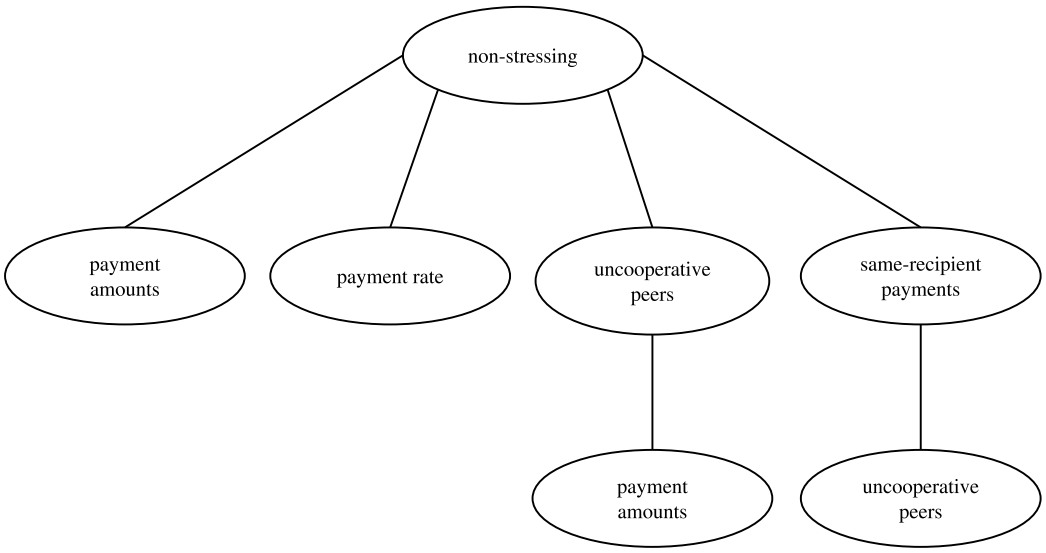


Figure 4.1: Branch-and-bound-like strategy for simulations on the LN mainnet.

Algorithm 1 shows the formal algorithmic description of such strategy, where $\mathcal{ST}$ represents the set of stressed independent variables and $P(\mathcal{ST})$ the probability of payment success resulting from the simulation with the stressed variables in $\mathcal{ST}$.

Algorithm 1 Branch-and-bound-like simulation strategy.

```
1: for each independent variable v do
2:     ST ← v
3:     for each independent variable u ≠ v do
4:         if P(ST) < 50% then
5:             break
6:         else
7:             ST ← ST + u
8:         end if
9:     end for
10: end for
```

4.2 Simulation Results

This Section presents the results of the simulations on the LN mainnet.
At the time of the snapshot, the LN mainnet showed the following features:

- Number of nodes: 1221.
- Number of channels: 5167.
- Average degree (number of open channels per node): 9.92.
- Average channel capacity: 381,350 satoshis.
- Gini index (concentration of bitcoins in channel capacities): 0.85.

As it can be noticed, the LN mainnet analyzed is in its early development and adoption stage. It is small, as few nodes and channels are present, and channel capacities are still low. Such premature state explains at least partially the non-operative cases found by the simulations.

The simulation results are presented branch by branch (see the branch-and-bound tree in Figure 4.1) to show the path followed to discover non-operative cases. For each discussed branch, the results of the simulations corresponding to that branch are showed, starting from the non-stressing simulation (the root of the tree), up to the last simulation in the branch (the leaf of the branch). The results

of each simulation are presented in the following way: the values of the independent variables in that simulation, underlining the stressed ones; the probability of payment success P_s (underlined when it is below 50%, as this corresponds to a non-operative case); the probability of payment failure for no route P_{f_r}; the probability of payment failure for unbalancing P_{f_b}; the probability of payment failure for uncooperative nodes $P_{f_{\bar{c}}}$. The results which include all output measures are showed in Appendix B.1, while the complete results including also variances and confidence intervals are available online[2]. Confidence intervals are not discussed in the following because, for what concerns probability of payment success and failures, those intervals resulted very strict, as it can be noticed in the online results: the largest difference between maximum and minimum values of the intervals was of the order of 0.01%.

4.2.1 Branch: Payment Amounts

In these simulations, the effects of stressing payment amounts was studied. Starting from the non-stressing simulation, a simulation was run in which σ_a was stressed. Table 4.4 shows the results.

Table 4.4: Simulation results of the branch of payment amounts.

Input Parameters				**Output Measures**			
r_π	σ_a	F_{sr}	$P_{\bar{c}_b}$	P_s	P_{f_r}	P_{f_b}	$P_{f_{\bar{c}}}$
10	1	0.0%	0.0%	65.43%	24.40%	10.11%	0.0%
10	<u>5</u>	0.0%	0.0%	<u>46.13%</u>	46.11%	7.72%	0.0%

A non-operative case was found, as with a stressed σ_a the probability of payment success resulted 46.13%, i.e., below 50%. Comparing the probabilities of payment failure between the non-stressing case and the amount-stressed case, an increase of the probability of payment failure for no route (46.11% against 24.40%) occurred, while the other probabilities of payment failure remained almost the same. This is due to the fact that capacities of channels are not enough to forward payments of higher amounts, thus a viable route for those payments cannot be found.

Further simulations were performed varying the value of σ_a within its variation interval (see Table 4.1), to investigate how it influences the network performance. The other input parameters were set to their non-stressing values. Table 4.5 shows the results of these simulations.

[2]https://researchdata.nexacenter.org/payment-network-simulator/LN-mainnet-results.zip

Table 4.5: Simulation results varying payment amounts.

Input Parameters				Output Measures			
r_π	σ_a	F_{sr}	$P_{\bar{c}_b}$	P_s	P_{f_r}	P_{f_b}	$P_{f_{\bar{c}}}$
10	5	0.0%	0.0%	46.13%	46.11%	7.72%	0.0%
10	4	0.0%	0.0%	50.77%	41.18%	8.00%	0.0%
10	3	0.0%	0.0%	56.87%	33.81%	9.30%	0.0%
10	2	0.0%	0.0%	63.35%	26.62%	9.96%	0.0%
10	1	0.0%	0.0%	65.43%	24.40%	10.11%	0.0%

With σ_a set to 4, the probability of payment success was higher than 50%, since, with respect to σ_a equal to 5, the probability of not finding a route decreased by 5%. Therefore, with σ_a equal to 4, the network is operative according to the definition of operativeness given in this Chapter.

In general, decreasing the value of σ_a caused an increase of payment success, due to the fact that low-amount payments more probably found a route with enough channel capacities. At the same time, a decrease of σ_a entailed a slight increase of payment failure for unbalancing, because some payments, although they found a route, failed for absence of sufficient balance in the route.

4.2.2 Branch: Payment Rate

In these simulations, the effects of a stressed payment rate was studied. Starting from the non-stressing simulation, a simulation with a stressed payment rate (100 payments per second) was performed. Table 4.6 shows the results of these simulations.

Table 4.6: Simulation results of the branch of payment rate.

Input Parameters				Output Measures			
r_π	σ_a	F_{sr}	$P_{\bar{c}_b}$	P_s	P_{f_r}	P_{f_b}	$P_{f_{\bar{c}}}$
10	1	0.0%	0.0%	65.43%	24.40%	10.11%	0.0%
100	1	0.0%	0.0%	43.88%	25.17%	30.92%	0.0%

A non-operative case was found, as the stressed payment rate caused a probability of success of 43.88%. The main reason is the increase of the probability of payment failure for unbalancing, which was around 20% higher with respect to the non-stressing simulation (30.92% against 10.11%). This is caused by the fact that channel balances were depleted by the increased number of payments to be forwarded.

4.2.3 Branch: Same-Recipient Payments

In this branch, starting from the non-stressing condition, first the fraction of payments directed to the same recipient was stressed, setting it to half the total number of payments (i.e., 5000 payments). The results in Table 4.7 show that such stress did not worsen performance. The probability of payment success increased with respect to the non-stressing simulation (69.45% against 64.43%). Such increase corresponds to a decrease of the probability of payment failure for no route: from 24.4% to 18.81%. The cause of this behavior is that the node to which half of payments were directed has many open channels. Consequently, payments directed to this node more probably found a viable route, with respect to the non-stressing case, in which payments were sent to different randomly-selected nodes.

To avoid to miss a non-operative case, a further simulation stressing a second parameter was ran. The only parameter that was possible to stress was the probability of uncooperative nodes. In fact, it was not possible to stress the payment amount, as the use case of payments directed to the same recipient is realistic only when the payments have low amount. And it was not possible to stress the payment rate, since it has been already showed that with 100 payments per second the network is non-operative. The results of this simulation show that, when both F_{sr} and $P_{\bar{c}_b}$ were stressed (the latter set to 10%), the LN mainnet remained operative. With respect to the case of only F_{sr} stressed, the probability of failures for uncooperative nodes increased to 14.84%, however, the probability of success was above 50% (57.1%), therefore the network resulted operative in this case too.

Table 4.7: Simulation results of the branch of same-recipient payments.

Input Parameters				**Output Measures**			
r_π	σ_a	F_{sr}	$P_{\bar{c}_b}$	P_s	P_{f_r}	P_{f_b}	$P_{f_{\bar{c}}}$
10	1	0.0%	0.0%	65.43%	24.40%	10.11%	0.0%
10	1	50%	0.0%	69.54%	18.81%	11.61%	0.0%
10	1	50%	10.0%	57.10%	18.80%	9.21%	14.84%

4.2.4 Branch: Uncooperative Nodes Probability

In this last branch of simulations, the effect of stressing the probability of uncooperative nodes was studied. The results in Table 4.8 show that, when the only stressed value was the probability of uncooperative nodes (set to 10%), the network resulted operative. In this case, the probability of payment failure for uncooperative nodes was 11.92%, but the probability of payment success remained above 50% (55.21%). The other probabilities of failure were almost the same as the non-stressing case.

Table 4.8: Simulation results of the branch of uncooperative nodes probability.

Input Parameters				Output Measures			
r_π	σ_a	F_{sr}	$P_{\bar{c}_b}$	P_s	P_{f_r}	P_{f_b}	$P_{f_{\bar{c}}}$
10	1	0.0%	0.0%	65.43%	24.40%	10.11%	0.0%
10	1	0.0%	10.0%	55.21%	24.31%	8.51%	11.92%
10	4	0.0%	10.0%	38.27%	46.10%	6.47%	9.13%

Again, to search for other non-operative cases, another parameter was stressed, i.e., the amount of payments (the payment rate was not stressed as it has already been showed that with 100 payments per second the network was non-operative, and the fraction of payments to the same recipient was not stressed because this case has already been studied in Section 4.2.3). To stress the payment amount, σ_a was set to 4 (not to 5, the highest possible level of payment amounts in the variation interval, because it has already been showed in Section 4.2.1 that with this value the network was not operative). The results of this additional simulation show that when stressing together the probability of uncooperative nodes and the payment amounts, the network resulted non-operative: the probability of success, in fact, was 38.27%. The probability of failure for uncooperative nodes was 9.14% (similar to the case in which the only stressed value was $P_{\bar{c}_b}$). However, the main reason of the failures is the absence of viable routes (with a probability of 46.1%), due to the increased amounts of payments.

Further simulations were performed to study which payment amounts are tolerated by the network when the probability of uncooperativeness is 10%. In those simulations, σ_a was varied, while $P_{\bar{c}_b}$ was fixed to 10%. Table 4.9 shows the results of these simulations.

Table 4.9: Simulation results varying payment amounts with 10% of uncooperativeness.

Input Parameters				Output Measures			
r_π	σ_a	F_{sr}	$P_{\bar{c}_b}$	P_s	P_{f_r}	P_{f_b}	$P_{f_{\bar{c}}}$
10	4	0.0%	10.0%	38.27%	46.10%	6.47%	9.13%
10	3	0.0%	10.0%	47.71%	33.78%	7.66%	10.82%
10	2	0.0%	10.0%	53.64%	26.56%	7.85%	11.91%
10	1	0.0%	10.0%	55.21%	24.31%	8.51%	11.92%

Comparing these results to the results of the simulations with zero uncooperativeness in Table 4.4, it can be noticed that in both cases, the lower the payment

amounts, the higher the probability of payment success. Moreover, when the probability of uncooperativeness was set to 10%, the network was operative with σ_a equal to 2, while with zero uncooperativeness, the network was operative also with higher payment amounts (σ_a equal to 4). The reason is that with $P_{\bar{c}_b}$ equal to 10%, around 9-11% of payments failed because of uncooperative nodes and the network reached probability of success higher than 50% only with σ_a lower than or equal to 2.

Finally, additional simulations were performed to study the level of uncooperativeness is tolerated by the network when σ_a is fixed to 4. Table 4.10 shows the results of the simulations varying the uncooperativeness.

Table 4.10: Simulation results varying uncooperativeness with σ_a equal to 4.

Input Parameters				**Output Measures**			
r_π	σ_a	F_{sr}	$P_{\bar{c}_b}$	P_s	P_{f_r}	P_{f_b}	$P_{f_{\bar{c}}}$
10	4	0.0%	10.0%	38.27%	46.10%	6.47%	9.13%
10	4	0.0%	1.0%	49.84%	41.17%	7.95%	0.99%
10	4	0.0%	0.1%	50.79%	41.15%	7.98%	0.06%

It can be noticed that only with the lowest value of uncooperativeness in the interval ($P_{\bar{c}_b}$ equal to 0.1%) the network resulted operative (probability of success is 50.79%). The reason is that, with a decrease of probability of uncooperativeness, there was a corresponding decrease of payment failures for uncooperative nodes (from 9.13% to 0.06%). However, the most considerable cause of failures was the absence of a viable route, because the payments of largest amounts did not find viable channels to be forwarded.

4.3 Answer to RQ1 and Main Findings

The Research Question 1 of this work is: "Which are the non-operative cases of the LN mainnet?". Basing on the simulation results presented in Section 4.2, the LN mainnet resulted non-operative in the following cases:

1. when the majority of the simulated payments was between 1 and 10^4 satoshis and the remaining payments (around 15%) were between 10^5 and 10^7 satoshis (namely, when σ_a was set to 5);

2. when the payment rate was set to 100 payments per second;

3. when the probability of uncooperative nodes was set to 10% and σ_a was set to 4.

In the following the above-mentioned cases are discussed, thus highlighting the main finding of the simulations in this Chapter.

Payment amounts When setting σ_a to 5, thus producing payments of the highest amounts in the defined interval, the LN mainnet resulted non-operative, since the probability of success was 46.13% (a payment is more likely to fail than to succeed). The main cause of the failures is the absence of viable payment routes, as the capacities of channels are not enough to forward payments of such amounts. While a naive solution would be to open channels with appropriate capacity, more elaborate approaches are beginning to appear, e.g., *Atomic Multi-Path Payments* [31], a technique that proposes to solve the payment amounts issue by splitting large payments into small ones which can be routed through channels with low capacity, and then recomposed by the receiver.

Payment rate When setting the average payment rate to 100 payments per second, the probability of success was 43.88%, so the LN mainnet resulted again non-operative. The main cause of such a high failure rate is channel unbalancing: channel balances were depleted by an increased rate of payments to be forwarded. Also in this case a naive solution would be to open channels with higher capacity, while a systemic alternative is represented by approaches akin to *REVIVE* [20], which allows channels to be re-balanced without the need of closing and re-opening them via on-chain transactions.

Uncooperative nodes and payment amount The last non-operative case resulted by the combination of a probability of uncooperative nodes set to 10% and σ_a set to 4. In this case, the probability of payment success was 46.1%. Payments most likely failed for absence of viable routes because of the high payment amounts. Moreover, with probability 9.14%, payments failed for uncooperative behavior of nodes. Such results leads to the following considerations. First, the probability of uncooperative nodes, as long as limited to the realistic value of 10%, does not constitute a serious problem for the network. Second, it has been proved once again that increasing payment amounts is the most critical issue for the LN configuration taken into account: a few recently proposed countermeasures were briefly mentioned above.

Chapter 5

Simulations on Synthetic Networks

This Chapters presents a set of simulations conducted on synthetic networks, i.e., HTLC networks generated by the network generator included in the CLoTH simulator (see Figure 3.2). The goal of these simulations was to study the impact of each individual input parameter on HTLC network performance. Therefore, this set of simulations took into account also the parameters defining the HTLC network, which were fixed in the simulations of the LN mainnet discussed in Chapter 4.

This Chapter is organized as follows. Section 5.1 discusses the design of the simulations on synthetic networks. Section 5.2 presents the simulation results. Finally, Section 5.3 discusses the main findings and answers Research Question 2.

5.1 Simulations Design

This Section discusses the independent variables and the strategy of the simulations conducted on synthetic networks.

5.1.1 Independent Variables

The independent variables of this set of simulations were all the simulator input parameters. Table 5.1 shows the variation intervals for each independent variable (a horizontal row separates network and payment parameters).

Table 5.1: Variation intervals of the independent variables.

Variable	Interval	Unity of Measure
N_n	[100,000 200,000 500,000 700,000 1,000,000]	-
N_{ch}	[3 5 8 11]	-
σ_t	[0 1 10 inf]	-
$P_{\bar{c}_b}$	[0 0.01 0.1 1.0 10.0]	%
$P_{\bar{c}_a}$	[0.01]	%
C_{ch}	[100 1,000 10,000 100,000 1,000,000]	satoshi
G	[0.0 0.1 0.2 0.4 0.6]	-
r_π	[10 100 1,000]	payments per second
N_π	[10,000 100,000 1,000,000]	-
σ_a	[1 2 3 4 5]	-
F_{sr}	[0 10 20 40 50]	%

The rationales of the variation interval (which were chosen after the analyzing some exploratory simulations) are explained in the following:

- Number of nodes N_n. A network with fewer than 100 thousands nodes is too small, since the LN is supposed to scale Bitcoin. With regard to the upper limit, as showed in Section 3.4, a simulation with 1 million nodes lasts more than four days. In future work the simulator performance will be improved and large networks will be analyzed.

- Average number of channels per node N_{ch}. The exploratory simulations showed that with fewer than three channels per node all payments fail, since the network is not properly connected. With 11 channels per payments, instead, no payments failed for absence of route, so performance would not improve with more than 11 channels.

- Topology tuner σ_t. Figure 5.1 shows the resulting topologies for each of the values of the variation interval. In this Figure, node size is directly proportional to the number of open channels of the node. It can be noticed that with σ_t= 0, there is just one single large node, which constitutes a centralized hub with many open channels. When σ_t= inf, instead, there are no hubs as all nodes have the same size. Finally, with the middle values of σ_t, there are some scattered hubs. Therefore, the two limits were chosen to produce the two opposite cases: totally decentralized topology and topology with one single centralize hub.

- Channel capacity C_{ch}. A channel capacity lower than 100 satoshis would

mean that the channel is unable to transfer anything but so-called dust payments. The upper limit was chosen in order to support the maximum payment amount and corresponds to one order of magnitude greater than such maximum.

- Gini index G. The Gini index is by definition a value between 0 and 1. The upper limit was chosen not too high to avoid that most of the network economic capacity is concentrated in very few channels.

The rationales of the remaining independent variables (namely, F_{sr}, N_π, r_π, G, $P_{\bar{c}_a}$, $P_{\bar{c}_b}$, σ_a) are the same of the simulations on the LN mainnet (see Section 4.1).

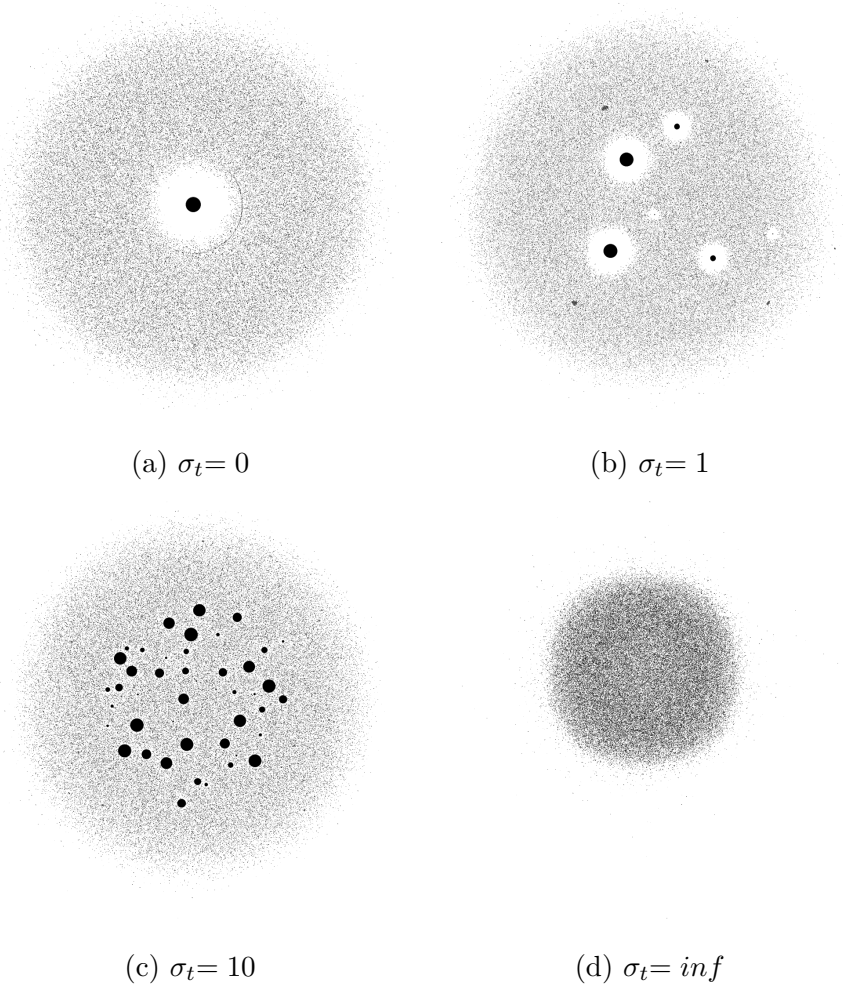
Table 5.2 shows the orders of magnitude of the payments generated for each value of the tuner. Notice that these orders of magnitude are decreased by one with respect to the simulations on the LN mainnet (cf. Table 4.2), because it has already been showed that higher orders of magnitude produce significant failures (see simulations results in Section 4.2.1).

Table 5.2: Ratio of payments with a certain order of magnitude for each value of σ_a.

Order of Magnitude (Satoshi)	$\boldsymbol{\sigma_a}$				
	1	**2**	**3**	**4**	**5**
10^{-1}	67.83%	35.26%	26.33%	20.38%	18.00%
10^{0}	27.61%	29.57%	23.76%	19.85%	17.43%
10^{1}	4.3%	18.80%	18.40%	17.14%	15.53%
10^{2}	2.6%	8.81%	13.79%	13.88%	14.14%
10^{3}	0.0%	3.32%	8.80%	11.48%	11.49%
10^{4}	0.0%	0.98%	5.08%	8.19%	9.69%
10^{5}	0.0%	0.20%	2.58%	5.55%	8.19%
10^{6}	0.0%	0.06%	1.26%	3.53%	5.51%

5.1.2 Simulation Strategy

As already stated, the goal of simulations on synthetic networks was to study the impact of each simulator input parameter on HTLC network performance. The strategy chosen consisted in studying the effect on performance of one independent variable at a time. For each independent variable, more simulations were performed, one for each of its values in the variation interval. When studying a variable, the variables not under observation were set to a *default value*, i.e., a reasonable value in the interval for which that variable does not interfere with the results of the simulation. For example, to study the effect of σ_t, four simulations were run, one



(a) σ_t= 0 (b) σ_t= 1

(c) σ_t= 10 (d) σ_t= inf

Figure 5.1: Resulting network topologies for different values of σ_t.

for each value of σ_t defined in the variation interval (see Table 4.1): in each one of the four simulations, the other independent variables are set to their respective default values.

In the following, the rationales of the chosen default values are explained (which Table 5.3 shows).

- Number of nodes N_n. A value higher than 100,000 was not chosen because of the limited maximum execution time of the simulator. However, a set of simulations varying the number of nodes in the variation interval was performed, in order to show the results for different values of such variable.
- Number of channels per node N_{ch}. The value suggested by the LN developers [40] was used.

- Network topology tuner σ_t. The chosen value produces no hubs in the topology, since the LN, as Bitcoin, is supposed to work in a decentralized topology.
- Uncooperative nodes probability $P_{\bar{c}_b}$ and $P_{\bar{c}_a}$. The chosen values represent a realistic probability that a node goes down for a fault.
- Channel capacity C_{ch}. The chosen value is higher than the maximum payment amount used in the simulations.
- Gini index G. The chosen value produces a uniform distribution of bitcoins in the channels. In this way, performance is not influenced by a non-uniform distribution of bitcoins, where some channels have an high capacity and the others have a low one.
- Average payment rate r_π. It is the middle of the variation interval.
- Number of payments N_π. It depends on the average payment rate, to ensure that the simulation does not last more than the maximum duration allowed (15 minutes).
- Payment amount tuner σ_a. It is set to the lowest level possible in its interval, to avoid the performance being influenced by high amount payments, when the effect of payment amount is not under observation.
- Fraction of same-recipient payments F_{sr}. It is set to zero to avoid that such payments affect performance in simulations in which their effect is not under observation.

Table 5.3: Default values of independent variables.

Variable	**Default Value**
N_n	100,000
N_{ch}	5
σ_t	inf
$P_{\bar{c}_b}$	0%
$P_{\bar{c}_a}$	0.01%
C_{ch}	100,000
G	0.0
r_π	100
N_π	100,000
σ_a	1
F_{sr}	0%

Differently from the simulations in Chapter 4, the branch-and-bound-like methodology was not adopted for finding non-operative cases of synthetic networks. In fact, with eleven input parameters, the branch-and-bound-like strategy would have required a very high number of simulations. This direction will be pursued in future work, once the simulator performance is improved.

5.2 Simulation Results

This Section presents the results of the simulations on synthetic networks. Since the simulations were performed varying one individual independent variable at a time, the results are presented separately for each independent variable under observation. In the following, only the most relevant results are showed, and confidence intervals are discussed only when they are large (indicating a relevant uncertainty in the results). The results including all simulations are available in Appendix B.2, while the results including also variances and confidence intervals are available online[1].

5.2.1 Channels per Node

In a decentralized network with 100 thousands nodes, three channels per node are not sufficient to have a robustly connected network. As Table 5.4 shows, with three channels, the probability of payment success was 59.61%; payments failed for absence of route (with probability 23.34%) and because the few existing channels became unbalanced (with probability 16.77%).

With five channels per node, instead, the performance was good: probability of success, in fact, was 99.34%. With 11 channels per node, the optimal condition was reached, as the probabilities of payment failure for no route and for unbalancing were zero (only a few payments failed or were delayed for uncooperative nodes).

Table 5.4: Simulation results varying the number of channels per node.

N_{ch}	P_s	P_{f_r}	P_{f_b}
3	59.61%	23.34%	16.77%
5	99.34%	0.31%	0.13%
8	99.82%	0.01%	0.0%
11	99.86%	0.0%	0.0%

[1] https://researchdata.nexacenter.org/payment-network-simulator/synthetics-results.zip

5.2.2 Uncooperative Nodes

As Table 5.5 shows, when the probability of uncooperative behavior of nodes was 10%, the probability of payment failure for uncooperative nodes was 11.84%. When $P_{\bar{c}_b}$ is lower, the network performance is not significantly affected by uncooperative behavior, as the probability of payment success was around 99%.

Table 5.5: Simulation results varying the uncooperative nodes probability.

$P_{\bar{c}_b}$	P_s	$P_{f_{\bar{c}}}$
0.0%	99.36%	0.0%
0.01%	99.34%	0.0%
0.1%	99.27%	0.1%
1%	98.32%	1.04%
10%	87.47%	11.84%

5.2.3 Network Topology

The average time to complete payments decreases as the network topology becomes more centralized. As proved by the results in Table 5.6, with one single hub (σ_t set to 0), average payment time was 333.16 ms, while it grew to 1391.92 ms in a totally decentralized network with zero hubs (σ_t set to infinite). This is caused by a corresponding decrease of the average length of routes traversed by payments: with one single hub, almost each node was connected to each other node through the hub, so a route was on average long 2.90 hops, while it reached 10.34 hops with zero hubs.

Table 5.6: Simulation results varying the network topology.

σ_t	P_s	T (ms)	L_r (hops)
0	99.88%	333.16	2.90
1	99.85%	536.83	4.13
10	99.83%	695.73	5.56
inf	99.34%	1391.92	10.34

However, also in a totally decentralized topology the synthetic HTLC payment network was characterised by good performance: the probability of payment success, in fact, was 99.34% and the average time to successfully complete payments was equal to 1391.91 ms, i.e., nearly instantaneous. However, in these simulations payment time was characterized by a small level of uncertainty, as Table 5.7 shows. The Table presents mean (μ), variance (σ^2), minimum and maximum values of

the confidence interval (C_{min} and C_{max}) of payment time, for different topologies. When the topology was totally decentralized, the average payment time actually ranged between 1343.93 and 1439.92 milliseconds.

Table 5.7: Mean, variance and confidence interval of payment time varying the network topology

σ_t	T (ms)			
	μ	σ^2	C_{min}	C_{max}
0	333.16	14.75	332.99	333.33
1	536.83	354.79	532.74	540.93
10	695.73	32.76	695.35	696.11
inf	1391.92	4155.77	1343.93	1439.92

5.2.4 Same-Recipient Payments

As Table 5.8 shows, when directing 50% of total payments (i.e., fifty thousands payments) to the same recipient, with a rate of 100 payments per second, the probability of failures for unbalancing reached 16.06%. The reason is that channels connecting to the destination node became unbalanced, as they were traversed always in the same direction. Such unbalancing did not constitute a problem if fewer payments are sent to the same recipient: already with F_{sr} set to 40%, the probability of payment failure for unbalancing decreased to 0.1%.

Table 5.8: Simulation results varying the fraction of same-recipient payments.

F_{sr}	P_{f_b}
0%	0.13%
10%	0.13%
20%	0.12%
40%	0.10%
50%	16.06%

5.2.5 Payment Amounts

Table 5.9 presents the simulation results obtained varying the payment amounts (see Table 5.2 for the mapping between σ_a and the resulting payment amounts).

Table 5.9: Simulation results varying payment amounts.

σ_a	P_s	P_{f_r}	P_{f_b}	T (ms)
1	99.34%	0.31%	0.13%	1391.92
2	99.13%	0.48%	0.19%	1427.54
3	96.80%	2.30%	0.65%	1611.93
4	93.69%	4.67%	1.33%	1816.37
5	91.43%	6.40%	1.85%	1949.87

Although the performance was good in all cases, the payment amounts have a significant impact on the results. In fact, when increasing the amounts, the probability of payment success decreased by about 8%. The main reason of failures was the absence of a viable route (with a probability of 6.4%). The cause is that payments of increased amount did not find channels with enough capacity to be forwarded.

Another factor that caused payment failures at the increase of payment amounts is channel unbalancing. With σ_a equal to 5, the probability of failures for unbalanced channel was 1.85%. This occurs because large payments depleted channel balances.

Finally, it is worth noticing that also the average payment time of successful payments increased at the increase of payment amounts. In fact, since channels became unbalanced, the payments bumping into an unbalanced channel were re-attempted and for this reason took more time to succeed. However, also in these simulations uncertainty characterized the results on payment time, as Table 5.10 shows. In fact, confidence intervals were larger and larger at the increase of σ_a. The reason is that payments with the lowest amounts took less time to complete than payments with largest amounts, which had to to be re-attempted multiple times.

Table 5.10: Mean, variance and confidence interval of payment time varying payment amounts

σ_a	T (ms)			
	μ	σ^2	C_{min}	C_{max}
1	1391.92	4155.77	1343.93	1439.92
2	1427.54	3708.31	1384.72	1470.35
3	1611.93	2122.15	1587.42	1636.44
4	1816.37	2917.32	1782.68	1850.06
5	1949.87	8370.85	1853.20	2046.54

5.2.6 Number of Payments

Table 5.11 shows the results of the simulations obtained varying the total number of payments executed during a simulation run (which lasts around 15 minutes).

It can be noticed that performance was good for each value of number of payments: the probability of payment success, in fact, is greater than 99.3% in each case. The probability of payments failing for no balance in the channel was low. The reason is that channel capacities are great enough to transfer payments of the default amount without experiencing unbalancing (see Table 5.3 for the default values of amounts and capacities).

Table 5.11: Simulation results varying the number of payments.

N_π	P_s	P_{f_b}
10,000	99.33%	0.19%
100,000	99.34%	0.13%
1,000,000	99.41%	0.07%

5.2.7 Number of Nodes

Table 5.12 shows the simulation results obtained varying the number of nodes, in a totally decentralized network without hubs.

Also in this case, the network showed a good performance, as the probability of payment success was over 99%. However, it can be noticed that the probability of failure for unbalancing slightly increased at the increase of the number of nodes. In fact, the larger the number of nodes, the lower the number of routes that connect each pair of peers in the network. As a consequence, if a payment did not find sufficient balance in a route, it may happen that an alternative route to the payment receiver was not found.

Also the average route length increased with the increase of the number nodes in the network. This is a consequence of the fact that, having more peers (randomly connected among them), the network is larger.

Table 5.12: Simulation results varying the number of nodes.

N_n	P_s	P_{f_b}	L_r (hops)
100,000	99.34%	0.13%	10.34
200,000	99.35%	0.11%	11.05
500,000	99.16%	0.22%	11.96
700,000	99.10%	0.26%	12.29
1,000,000	99.06%	0.33%	12.63

5.2.8 Channel Capacity

Table 5.13 presents the results of the simulations varying the per-channel capacity. It is important to highlight that in these simulations payment amounts were fixed to the lowest level possible (i.e., σ_a= 1).

The payment network performance was good in all the cases, being the probability of payment success always around 99%. The simulation with the worst performance (P_s= 98.37%) is the one with the lowest value of per-channel capacity. In that case, the majority of payment failures was caused by channel unbalancing, and the rest by the absence of routes with sufficient channel capacities. In addition, the average payment time reached around 2 seconds per payment: in fact, with the lowest value of channel capacities, channels were subject to unbalancing, and payments had to be re-attempted multiple times before succeeding.

Table 5.13: Simulation results varying channel capacity.

C_{ch} (satoshi)	P_s	P_{f_r}	P_{f_b}	T (ms)
100	98.37	0.51	0.81	2134.47
1,000	99.36	0.32	0.13	1404.00
10,000	99.35	0.31	0.13	1392.36
100,000	99.34	0.31	0.13	1391.92
1,000,000	99.34	0.31	0.13	1391.80

5.2.9 Gini Index

Table 5.14 shows the simulation results produced varying the Gini index. The variation of the index did not cause changes in the probability of payment success. The reason is that the average channel capacity (100,000 satoshis) was high enough for the payment amounts simulated, and even a non-uniform distribution of the capacity did not produce significantly low-capacity channels.

Table 5.14: Simulation results varying the Gini index.

G	P_s
0.0	99.34
0.1	99.34
0.2	99.34
0.4	99.34
0.6	99.34

5.3 Answer to RQ2 and Main Findings

The Research Question 2 of this work is: "Which is the impact of the simulator input parameters on payment networks performance?". Basing on the simulation results discussed in this Chapter, the following answer is provided, for each simulator input parameter:

- Channels per node N_{ch}. The lower the number of channels per node, the higher the probabilities of failures for no route and for unbalancing. However, with 5 channels per node or more, the probability of payment success was greater than 99%.
- Uncooperative nodes probability $P_{\bar{c}_b}$. Uncooperative nodes cause payment failures, but the impact is not particularly significant: in fact, only in the worst case simulated, when the probability for a node to be uncooperative was 10%, the payment success rate went below 99%.
- Network topology σ_t. A centralized topology reduces the average payment time and the average route length with respect to a totally decentralized topology. The probability of payment success, instead, resulted greater than 99% in both a centralized and a decentralized network.
- Same-recipient payments F_{sr}. The percentage of same-recipient payments influences the probability of failures for channel unbalancing. Such impact is significant only when the percentage is 50%, which causes around 16% of failure for unbalancing.
- Payment amounts σ_a. An increase of payment amounts cause increases of failures for absence of route and for unbalancing, and an increase of the average payment time. In the worst case, with σ_a set to 5, the probability of payment success is around 91%.
- Number of payments N_π. An increasing number of payments does not cause any significant effect on the network performance.
- Number of nodes N_n. The number of nodes has a slight influence on the probability of failures for unbalancing and on the average route length, which both increase at the increase of nodes.
- Channel capacity C_{ch}. The average channel capacity has a contained impact on the failures for no route and for unbalancing, and also on the payment mean time, which slightly increase when channel capacities decrease.
- Gini index G. The Gini index, which defines the distribution of capacity in the channels, has no significant effect on the network performance.

Finally, the main findings of the simulations on synthetic networks can be summarized as follows.

Synthetic networks performance In general, synthetic networks perform well: the probability of payment success, in fact, was close to 99% in almost all cases. Therefore, it has been proved that the values of the network parameters used in this set of simulations (e.g., the average channel capacity and the number of channels per node) are good candidates as ideal reference values that network participants should actively try to maintain in a distributed effort to keep the network in good health.

Channels per node In a decentralized network with 100 thousands nodes, where each node has only three active channels, a payment succeeded with probability 59.61%, i.e., much less than the 99% that can be reached with five active channels per node. Therefore, at least five channels per node are needed to have a robustly connected network.

Uncooperative nodes If a node is uncooperative once every 10 times, the probability for a payment to fail due to such uncooperative behavior was 11.83%. Hence, the probability of uncooperative nodes (unless unrealistically high) does not constitute a serious issue in the synthetic networks analyzed.

Network topology The average payment time in a totally decentralized topology was approximately 1.4 seconds, i.e., nearly instantaneous. As to be expected, the presence of well-connected hubs acting as payment gateways further reduced the average payment time.

Payment amounts The higher the payment amounts, the higher the probability of failure for absence of route and for channel unbalancing. With the highest level of amounts (see Table 5.2 at $\sigma_a = 5$) the probability of payment success went down to 91.43%.

Same-recipient payments Thousands of payments directed toward the same recipient in a short time window determined the unbalancing of channels involved, especially those closer to the payee. Merchants who expect to receive a significant payment throughput should open many payment channels (more than the default five), to prevent payment losses.

Chapter 6

Network and Protocol Modifications on the Lightning Network Mainnet

This Chapter presents simulations conducted on the LN Mainnet whose goal is to study the network performance when certain network and protocol modifications are applied.

The protocol modifications analyzed are rebalancing approaches. They are implemented to tackle the channel unbalancing, which turned out to be an issue from the simulations results of Chapter 4.

The network modifications analyzed are of two separate types: removal of hubs from the network; definition of classes of nodes (payees-only, payers-only, hybrids) in order to simulate the typical scenario of few service-providers nodes that receive payments for their services from the remaining nodes of the network.

All the simulation results presented in this Chapter are plotted with error bars: the errors are computed using 95% confidence intervals of the mean values. Only the most relevant results are discussed. The complete results are showed in Appendix B.3.

This Chapter is organized as follows. Section 6.1 presents the LN mainnet taken as reference for the simulations of this Chapter: the main features of this network are showed and exploratory simulations conducted on this network are discussed. Section 6.2 presents the simulations conducted when hubs are removed from the LN mainnet. Section 6.3 discusses the rebalancing approaches and the results produced when simulating these approaches on the LN mainnet. Section 6.4 presents the simulations on the LN mainnet with the classes of nodes that define the service-providers scenario. Finally, Section 6.5 answers to Research Question 3 and discusses the main findings.

6.1 The LN Mainnet

In all the simulations discussed in this Chapter, a snapshot[1] of the LN mainnet was analyzed. This Section presents the main features of the LN mainnet at this snapshot (Section 6.1.1), exploratory simulations conducted on an optimal configuration for the LN mainnet (Section 6.1.2) and exploratory simulations conducted on the LN mainnet (Section 6.1.3).

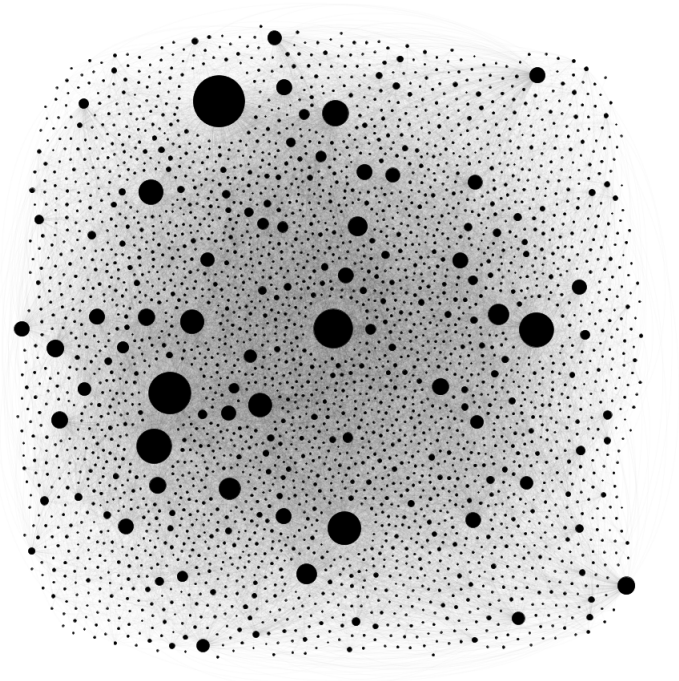


Figure 6.1: The LN mainnet on February 12th 2019.

6.1.1 Features of the Network

Figure 6.1 is a graphical view of the LN mainnet, where size of nodes is directly proportional to their number of open channels. The LN presents the following features:

- Number of nodes with at least one open channel: 3148;

[1]Downloaded on February 12nd 2019 at 16:36 CET from https://rompert.com/recksplorer/

- Number of channels: 24683;
- Mean and standard deviation of channel capacity: 0.0267 BTC (mean), 0.04361 BTC (standard deviation).

With respect to the older snapshot on which the simulations described in Chapter 4 were conducted (see Section 4.2), the LN mainnet has grown and the following can be noticed: an increase of 157% of the number of nodes, an increase of 377% of the number of channels and an increase of 600% of the mean channel capacity.

Table 6.1 shows the channel policies in the mainnet. For each policy, the table shows also default value set when opening a channel with the `lnd` client.

Table 6.1: Channel policies in the LN mainnet.

Policy	**Mean**	**STD**	**Default value**
Minimum HTLC (msat)	1034.79	40300.17	1000
Base fee (msat)	959.58	2504.26	1000
Proportional fee (msat)	671.66	22173.42	1
Timelock (blocks)	134.75	68.29	144

6.1.2 Optimal Configuration

In this Section a set of exploratory simulations are presented, whose aim was to study the LN mainnet behavior in an optimal configuration for the network.

The optimal configuration is defined in the following way:

- The capacity of each LN mainnet channel is increased to 3 BTC.
- All channels are perfectly balanced (50% of the channel capacity for each channel party).
- Only payments that are several orders of magnitude lower than the channel capacities are simulated on such modified LN mainnet - specifically, payments uniformly distributed between 1 and 10^5 satoshi.
- All nodes are always cooperative.

The behavior of the LN mainnet in the above-defined optimal configuration was studied at different payment rates: Figure 6.2 shows the results of the simulations in the optimal configuration. In particular, the Figure shows the probability of payment success resulting from different payment rates. In addition, the optimal configuration was compared with the original LN mainnet (with the original channel capacities not increased to 3 BTC); the same payments of the optimal case were simulated on the original mainnet.

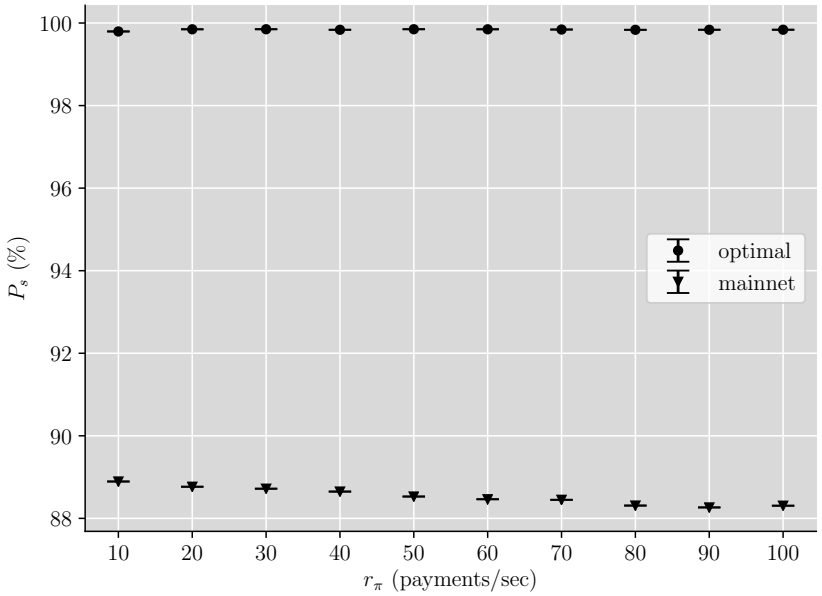


Figure 6.2: Probability of payment success in the optimal configuration and in the original LN mainnet.

With regard to the optimal configuration, as expected, the probability of payment success was high: around 99.8% for each payment rate. As the complete results in Appendix B.3 show (Table B.11), payments failed only for the absence of a route. From an analysis of the raw-per-payment data produced by the simulator, it was possible to notice that such payments failed in finding a route either because there is no connection in the graph between the payment sender and the payment receiver, or because they do not respect the minimum HTLC policy.

Instead, with regard to the simulations on the original mainnet, payment success probability was around 10% lower than the optimal configuration. The reason of failures are the absence of a route (around 8-9%) and channel unbalancing (1-2%). The cause of such performance decline is that channel capacities of the original LN mainnet are lower with respect to the capacities in the optimal configuration: payments do not find a route with sufficient channel capacities, and low-capacity channels are subject to unbalancing.

6.1.3 Simulation Results

This Section presents exploratory simulations conducted on the LN mainnet with the goal of defining the variation intervals of the independent variables.

Independent Variables The only independent variable considered for the simulations of this Chapter was the payment amounts tuner σ_a. The other simulator input parameters were not considered as independent variables for the following reasons:

- The network parameters are defined by the LN mainnet.
- The parameters concerning the probability of uncooperative nodes were fixed to zero, since the uncooperative behavior of nodes was not under observation in the simulations of this Chapter.
- The percentage of same-recipient payment was fixed to zero, since the network behavior in the scenario of payments directed to only few nodes was studied by simulations in Section 6.4.
- As Figure 6.2 shows, the payment success probability of the LN mainnet did not significantly vary for different payment rates. Therefore, a fixed payment rate was used. The value chosen is 100 payments per second, because it constitutes the middle order of magnitude between 10 payments per second - the Bitcoin transaction rate which the Lightning Network is supposed to overcome - and 1000 payments per second - the payment rate of well-established payment systems that the Lightning Network aspires to reach in the future.

Payment amounts variation interval The orders of magnitude of payment amounts were chosen between 10^0 and 10^5 satoshis: the minimum was chosen because the default minimum HTLC policy is 1 satoshi, therefore payments lower than 1 satoshi would almost certainly fail; the maximum was chosen to avoid to produce payments higher than the average channel capacity of the LN mainnet.

The amount tuner σ_a sets the distribution of payment amounts in the above-defined orders of magnitude. To define the limits of the variation interval of σ_a, the following procedure was adopted: the lower limit of σ_a was set to 1, because this value produces almost all payments of the lowest amount possible (1 satoshi); the upper limit was chosen via simulations on the LN mainnet. In these simulations, σ_a was increased until the probability of payment success went below 90%. This is to avoid to use values of σ_a for which it is already known that they cause low performance. Table 6.2 shows the distribution of payment amounts for each value of the variation interval of σ_a defined as above.

Simulation results Figure 6.3 shows the results of the above-discussed simulations on the LN mainnet varying σ_a. The probability of payment success was below 90% for σ_a= 8.0, so this value was chosen as upper limit of the variation interval. When σ_a= 1, the probability of payment success was 99.81% and the only reason of payment failures was for absence of route. While increasing σ_a, the failures for

Table 6.2: Ratio of payments with a certain order of magnitude for each value of σ_a.

Order of Magnitude (Satoshi)	σ_a							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
10^0	97.44%	65.58%	42.93%	31.42%	26.64%	22.74%	21.12%	20.13%
10^1	2.55%	25.41%	26.97%	24.92%	21.86%	21.16%	19.98%	19.61%
10^2	0.01%	7.05%	16.15%	18.32%	18.50%	17.81%	17.58%	17.61%
10^3	0.0%	1.63%	8.61%	12.29%	15.19%	15.58%	16.08%	15.54%
10^4	0.0%	0.28%	3.82%	8.01%	10.16%	12.90%	13.50%	14.21%
10^5	0.0%	0.05%	1.52%	5.04%	7.65%	9.81%	11.74%	12.90%

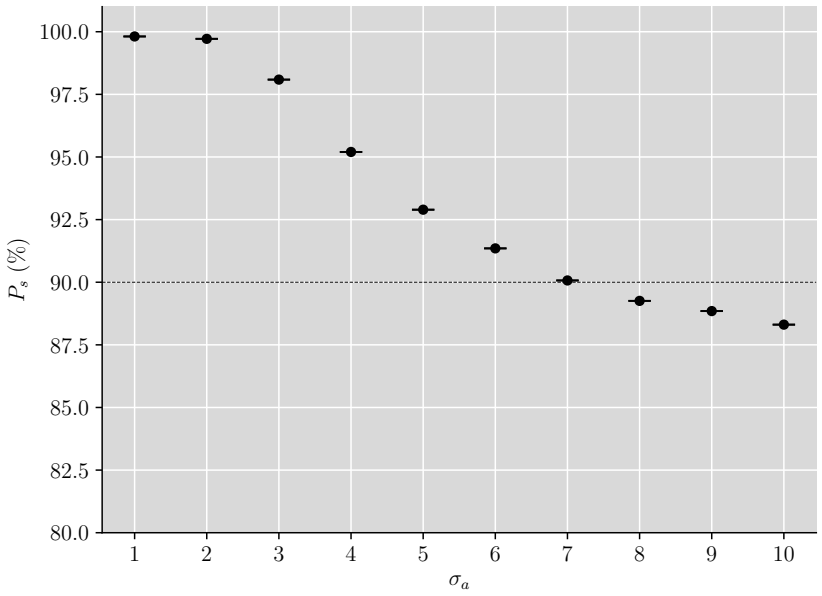


Figure 6.3: Payment success probability for each σ_a in the LN mainnet.

no route increased and some failures were also due to unbalancing and to payment timeout expiration. In general, the most occurring reason for failures was the absence of route (around 8%), while channel unbalancing was the cause of around 2% for the highest values of σ_a analyzed.

As already discussed in other simulations, the reason of this behavior is that payments with increasing amounts did not find a route with sufficient channel capacities, and also caused channel unbalancing.

Other interesting results coming from the simulations on the LN mainnet are the average payment time and the average payment route length, showed in Appendix B.3 (Table B.12). With regard to the first, it increased with the increase of payment amounts (because channels unbalance and payment needs to be re-attempted), but it is never higher than 525 milliseconds. The average payment route length is always around 3.3 hops.

6.2 Hubs

This Section discusses a set of simulations which analyzed the LN mainnet performance when hubs of the network were removed. The goal was to understand how hubs influence the LN mainnet.

Identification of hub In network science, a hub is a node characterized by a high degree [2], i.e. a high number of connections. In this work, the first six nodes of the LN mainnet ordered by number of channels are considered as hubs.

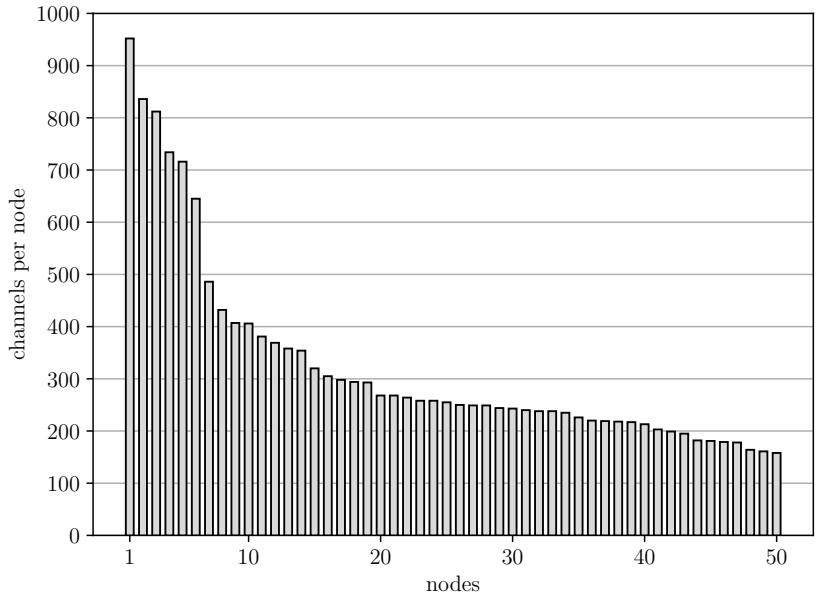


Figure 6.4: Number of channels per node of the first 50 most connected LN nodes.

Figure 6.4 is the bar plot of the number of channels per node in decreasing order for the first 50 most connected nodes. The reason of choosing the first 6 nodes as hubs is justified by the fact that there is the highest step between the sixth and the seventh node: the seventh node has 159 less channels than the sixth. The chosen hubs have in total 4695 channels, that constitute around 20% of the total number of channels in the LN mainnet.

Simulation strategy The simulation strategy adopted for studying the influence of hubs in the network consisted in disconnecting one-by-one the hubs and in running a simulation on the resulting network without those hubs. From the LN mainnet, the first hub (the one having the highest number of channels) was removed with all its channels and a simulation was performed on the network without that hub. Then, from the network without the first hub, the second hub was removed

and a simulation was run on the resulting network (namely, the LN mainnet lacking of the first and second hub). The same process was repeated until all hubs were removed.

The only independent variable of these simulations was the number of disconnected hubs, as the objective of the simulation was to analyze their impact on performance. The payment amount parameter σ_a was fixed to 4, as it represents the middle of its variation interval defined in Section 6.1.3 .

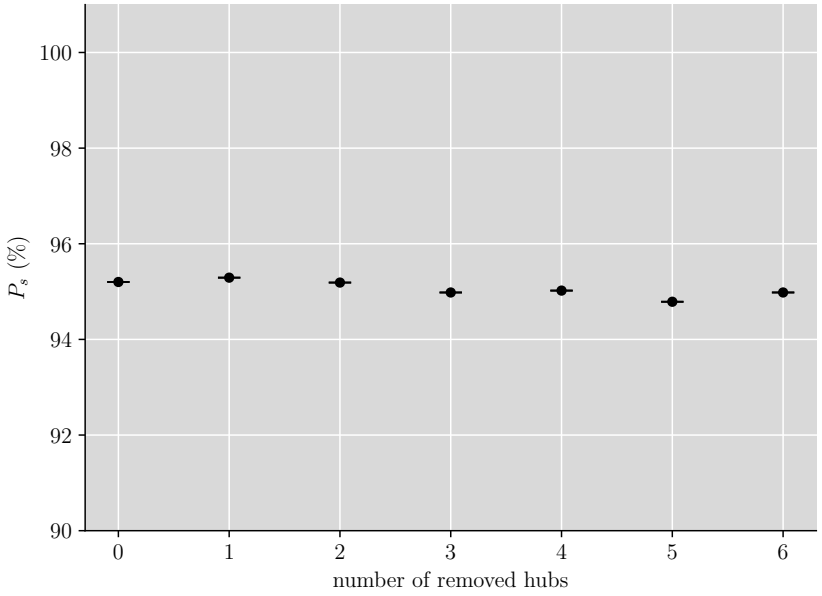


Figure 6.5: Payment success probability when removing hubs in the LN mainnet.

6.2.1 Simulation Results

Figure 6.5 shows the probability of payment success when varying the number of hubs removed from the LN mainnet. As already mentioned, in each simulation σ_a was fixed to 4 (see Table 6.2 for the corresponding distribution of payment amounts).

The payment success rate is not significantly affected by the presence of hubs: when all the hubs and their channels were connected, such rate was 95.2%, while when all hubs were disconnected, it was 94.98%, that means a decrease of around 0.2%. Such decrease was entirely caused by payments that did not find a route.

The most visible effect resulting from hubs removal was a slight increase of the average route length, as Figure 6.6 shows: from 3.34 to 3.62 number of hops.

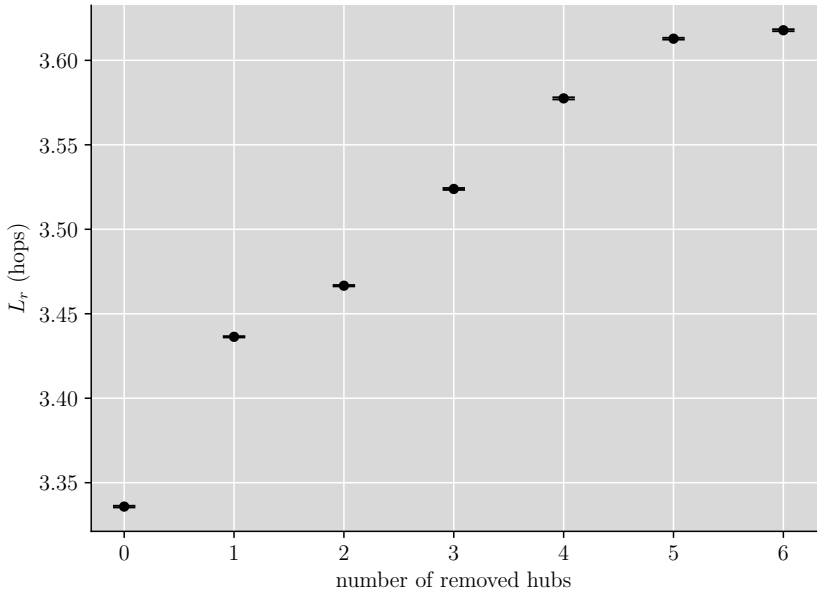


Figure 6.6: Average payment route length when removing hubs in the LN mainnet.

6.3 Rebalancing Approaches

In this Section, two different rebalancing approaches are described, whose objective is to rebalance channels, thus avoiding that payments fail because of unbalanced channels. After implementing the rebalancing approaches in the simulator, some simulations were performed to understand whether the approaches are effective, namely, whether they succeed in reducing the probability of failures for unbalancing. Two different rebalancing approaches has been designed in this work: active and passive rebalancing.

Active rebalancing The active rebalancing consists in executing a payment with the purpose of rebalancing channels. If a node has a channel with low balance and also another channel with high balance, it executes a rebalancing payment, which transfers funds from its high-balance channel to its low-balance channel.

Consider the situation in Figure 6.7 . In the channel with Carola, Alice has a low balance (10% of the total channel capacity), while in the channel with Berto she has a high balance (90% of the total channel capacity). Therefore, she performs a rebalancing payment that moves 0.4 BTC from the channel with Berto to the channel with Carola. In this way, the channel with Carola will result balanced, as she and Carola will own both 50% of the total channel capacity.

The active rebalancing implemented in the simulator is triggered when a channel balance goes below 20% of the total channel capacity. The amount transferred with

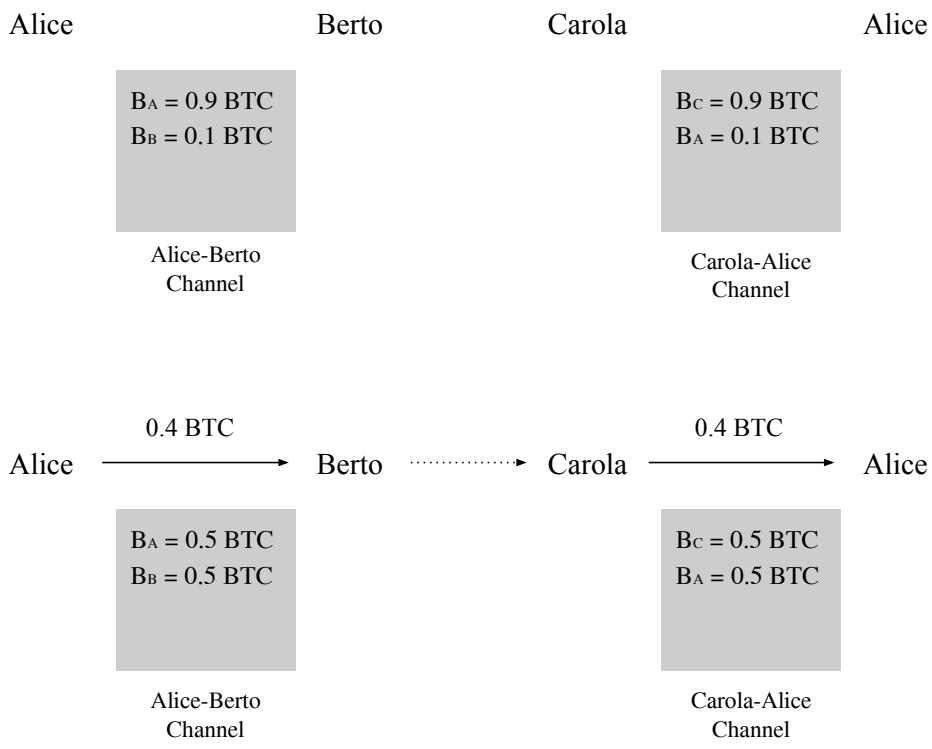


Figure 6.7: Rebalancing payment in the active rebalancing approach.

the rebalancing payment is the amount necessary to increase the channel balance up to the 50% of the channel capacity.

The active rebalancing approach fails in the following cases:

- When the node with a low-balance channel does not have any candidate channel for rebalancing. The candidate channel must satisfy two conditions: (i) the node's balance in that channel must be greater than half of the total channel capacity; (ii) the node's balance in that channel must be sufficient to transfer the rebalancing payment amount.
- When the rebalancing payment fails for no route (in the example of Figure 6.7, when there is no route between Berto and Carola).
- When the rebalancing payment fails because there is not enough balance in the channels of the found route (in the example of Figure 6.7, when one or more channels in the route between Berto and Carola have a balance lower than 0.4 BTC).

For the sake of simplicity, it is assumed that rebalancing payments are executed instantly and no fees are charged to them.

Passive rebalancing The passive rebalancing consists in adjusting the fee policy of a channel according to the channel balance, in a way that fee amount is inversely proportional to channel balance. The rationale of this approach is to encourage payments to traverse channels with high balances. In fact, the Dijkstra's algorithm used for finding a route for a payment tends to prefer routes with lower fees: therefore, if fee is kept inversely proportional to balance, payments should tend to traverse channels with high balances and to avoid channels with low balances. A linear function is used to map channel balance to channel fee policy.

Simulation strategy To study the effectiveness of the rebalancing approaches discussed, some simulations were performed using a version of the simulator that implements the approach under observation. For each rebalancing approach, multiple simulations were conducted varying σ_a. The LN mainnet described in Section 6.1 was given in input to the simulator in these simulations.

6.3.1 Simulation Results

Active rebalancing Figure 6.8 shows the probability of payment failures for unbalancing when active rebalancing is implemented, compared with the case in which no rebalancing approach is implemented.

As it can be noticed, the difference of the two cases is slight: the probability of payment failures were almost the same in both cases.

The low effectiveness of the active rebalancing is caused by the fact that most of the attempts of rebalancing did fail. Table 6.3 shows, for each value of the payment amount tuner σ_a, the total number of active rebalancing attempts, the percentage of succeeded and the percentages of failed for each of the three causes explained above (absence of a candidate channel for unbalancing, failure of the rebalancing payment for no route, failure of the rebalancing payment for no balance). The Table shows that for any σ_a, just a few rebalancing attempts succeeded: at maximum around 16%. The majority of attempts failed because there was not enough balance to transfer the rebalancing payments. A substantial part also failed because a candidate channel from which executing the rebalancing payment was not found.

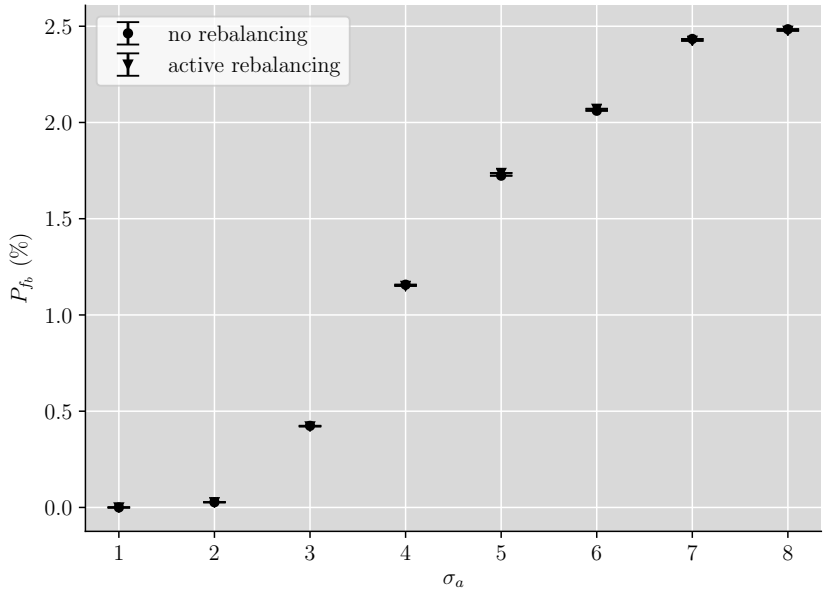


Figure 6.8: Probability of payment failures for unbalancing with/without active rebalancing in the LN mainnet.

Table 6.3: Rebalancing attempts.

σ_a	Attempts	Succeeded (%)	No channel (%)	No route (%)	No balance (%)
1	1726	4.06	20.74	2.84	72.36
2	3644	9.03	27.72	2.00	61.25
3	22067	11.48	21.52	2.56	64.44
4	36032	12.89	25.85	2.81	58.44
5	44819	16.32	23.11	2.08	58.49
6	52079	16.05	24.71	2.16	57.07
7	52557	15.52	27.16	2.47	54.85
8	57013	15.87	27.51	2.64	53.99

From that follows that the active rebalancing approach implemented is not effective for two reasons: mainly because channels with not sufficient balance in the network cause failures of the rebalancing payments; secondarily, because a candidate channel for executing the rebalancing payment is not found.

Passive rebalancing Figure 6.9 shows the probabilities of failures for unbalancing with and without the passive rebalancing approach. Passive rebalancing is more effective than active rebalancing: the probability of failures for unbalancing was lower than the case without rebalancing for any value of σ_a.In particular, failures for unbalancing decreased of about one fourth in each case when passive rebalancing is implemented.

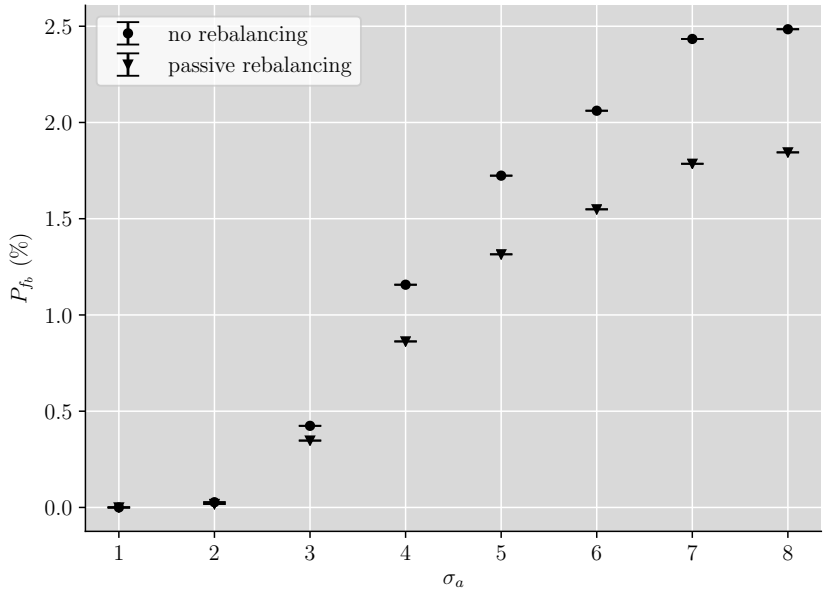


Figure 6.9: Probability of payment failures for unbalancing with/without passive rebalancing in the LN mainnet.

Passive rebalancing causes also a slight increase of failures for no route, as results in Appendix B.3 show (Table B.15). The reason for such increase is that certain payments had to pay higher fees than in the case without passive rebalancing: these payments failed when there were not sufficient capacities in the channels for transferring the payment amounts plus the increased fees. However, the decrease of failures for no balance was greater with respect to the increase of failures for no route, hence passive rebalancing remains effective.

6.4 Service-Providers Scenario

In this Section the LN mainnet was studied in the service-providers scenario. The service-providers scenario is a typical case of use of the Lightning Network, in which a few nodes in the network (service providers) are payees only, as they are paid for their services. The majority of network nodes, instead, are payers only, as they send payments to the service providers.

The goal of the simulations in the service-providers scenario was to understand whether the LN mainnet can support this typical case of use in which most of the payments are directed only to a few service-providers nodes.

Classes of nodes To configure the scenario, three classes of nodes were set in the LN mainnet:

- Payees: they represent the service providers, namely, nodes that only receive payments. They are nodes directly connected to one of the six hubs of the LN mainnet (according to the identification of hub made in Section 6.2). The reason of this choice is that it is convenient for a service provider to be connected to a hub, as also many other nodes are connected to the hub and therefore their payments can easily reach the provider. The service providers are expected to be a minority, so the payees are 10% of the nodes directly connected to a hub. In addition, each payee was chosen in a way that, in the channel with the hub, the hub's balance is higher than the payee's balance, to avoid that payments flowing through the hub directed to the payee fail for not sufficient balance. Payees are 188 nodes in total.
- Payers: they are the nodes than only send payments. For the same reason of having an efficient connection to the service providers, also payers are directly connected to a hub. Therefore, they constitute the remaining nodes connected to a hub which are not payers: in total, they are 1781.
- Hybrid nodes: they are nodes that both send and receive payments. They are the remaining nodes of the LN mainnet which do not belong to the categories of payees and payers: they are 1179 nodes in total.

In the simulations on the LN mainnet where the above-defined classes of nodes were set, 80% of the simulated payments flowed from payers to payees, while the rest was exchanged among hybrid nodes, in order to preserve a certain level of traffic in the network not directed to the service providers.

Simulation strategy To study the performance of the LN mainnet in the service-providers scenario, multiple simulations were run varying σ_a in the defined interval. The LN mainnet with the above-defined classes of node was given in input to the simulator in each simulation.

6.4.1 Simulation Results

Figure 6.10 shows the probabilities of payment success of the simulations in the service-providers scenario, comparing it to the normal scenario (where no classes of nodes were set in the LN mainnet).

It can be noticed that, as payment amounts increased, the success rate in the service-provider scenario decreased, up to around 66%. The main reason of payment failures is channel unbalancing, which in the worst case (in correspondence to the highest σ_a) reached a probability of around 26% (see results in Appendix B.3, Table

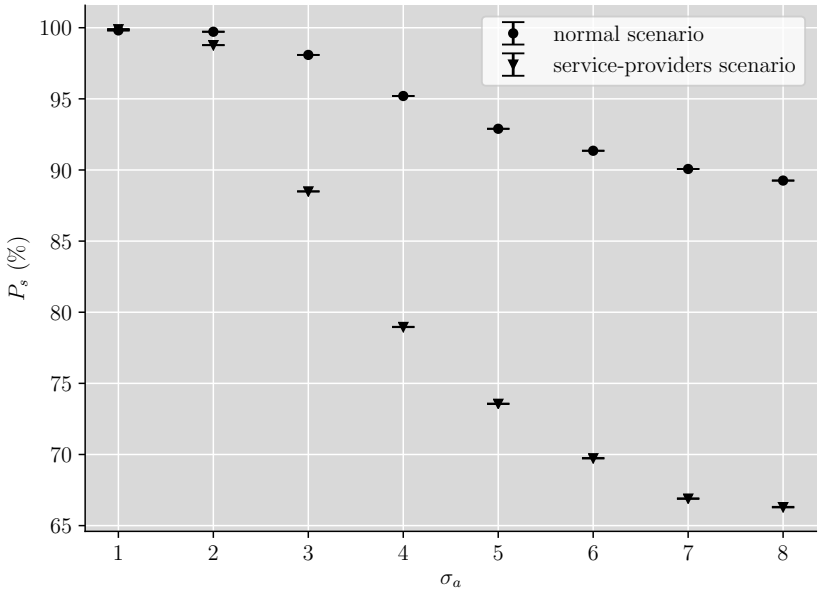


Figure 6.10: Payment success probability in the service-providers scenario in the LN mainnet.

B.16). Such high probability of failures is due to the fact that, since payments were mostly directed to a few nodes, channels connecting to the nodes became unbalanced.

For this same reason, it can be noticed an increase of average payment attempts necessary before a payment succeeded, as Figure 6.11 shows. In fact, since payments bumped into unbalanced channels, they had to be re-attempted more and more times. In addition, it can be noticed that error bars increased at the increase of σ_a. The cause is that payments of lowest amounts required less attempts to be completed with respect to payments with highest amounts.

6.5 Answer to RQ3 and Main Findings

Research Question 3 of this work is: "How do network and protocol modifications affect performance of the LN mainnet?". According to the simulation results discussed in this Chapter, the following answer to RQ3 is provided:

- Protocol modifications. Two different rebalancing approaches were analyzed to tackle channel unbalancing: active rebalancing, which consists in executing payments that rebalance channels; passive rebalancing, which consists in adjusting fees to direct payments toward balanced channels. The active

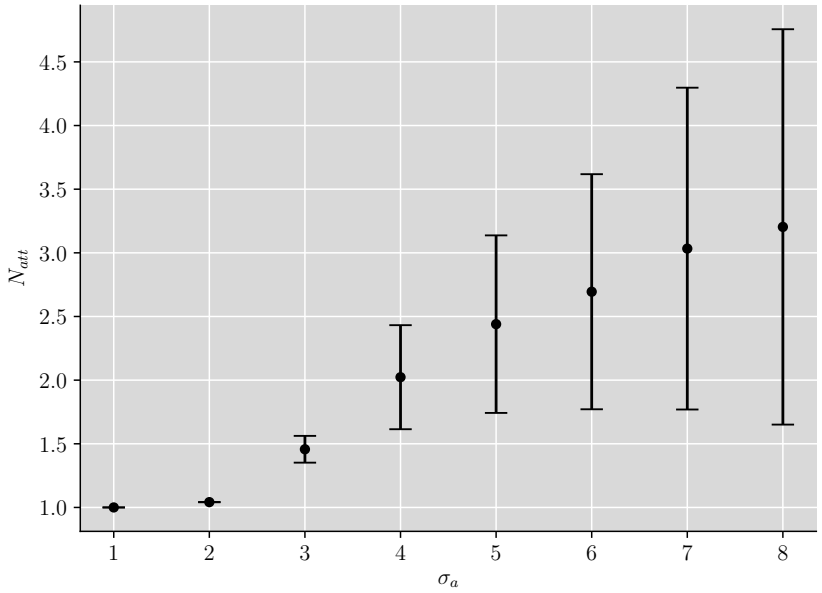


Figure 6.11: Average payment attempts in the service-providers scenario in the LN mainnet.

rebalancing had an irrelevant impact on rebalancing, as most of the rebalancing payments failed, mainly because there were not sufficient balance in the channels of the routes traversed by the rebalancing payments. The passive rebalancing, instead, was effective, as it reduced the failures for unbalanced channels of about one fourth with respect to the case in which no rebalancing approach is implemented.

- Network modifications. The network modifications applied to the LN mainnet were: removal of hubs from the network; definition of classes of nodes (payee, payers and hybrid) to represent the service-providers scenario. For what concerns hubs, disconnecting them from the network did not have a significant impact on the performance, as the probability of payment success decreased by 0.2% when all hubs were removed. With regard to the service-providers scenario, a noticeable part of payments failed for unbalancing (around 26% when the highest level of payment amounts were simulated), as channels directing to the payees became unbalanced.

In the following, the main findings of the simulations described in this Chapter are discussed.

Rebalancing The passive rebalacing strategy which adapts fees to channel balance may be a promising solution to address channel unbalancing. In the future

work, some improvements of the approach can be explored. For example, a non-linear function can be used for fee-balance mapping, which is characterized by a steeper slope and sets decreasing fees as a channel balance increases.

The version of active rebalancing implemented does not significantly reduce unbalancing. Since most of the rebalancing payments failed for not sufficient balance in the channels of their route, a possible solution could be to re-attempt the rebalancing payments multiple times, until a suitable route with enough balances is found.

Hubs The LN mainnet results resilient to disconnection of hubs: even when the first six nodes with the highest number of channels (that together constitute the 20% of the total channels) were removed from the network, the probability of payment success did not significantly decrease.

However, in this investigation the hubs considered were only the ones having an high number of channels. In future work, eigenvector centrality and other centrality measures will be used to identify the most central network nodes. To study their influence on the network performance, simulations will be run removing the identified central nodes.

Service-providers scenario In the service-providers scenario, channel unbalancing causes a significant amount of payment failures. In the future work, the passive rebalancing approach discussed in this Chapter can be implemented in the service-providers scenario, to understand whether it succeeds in reducing the failures. In general, to address this issue, service-provider nodes, which are supposed to receive many payments, are encouraged to open as many channels as possible.

Chapter 7

Conclusions and Future Work

Payment channel networks are the most explored solution to the largely discussed issue of blockchain scalability. They are networks of payment channels which enable off-blockchain payments. The Lightning Network is the mainstream and most developed payment channel network, built on top of the Bitcoin blockchain. It leverages the HTLC contract to execute off-chain payments. The Lightning Network is in its early stage of development and presents critical features that are worth being thoroughly studied, such as the limit imposed by channel capacities to payment amounts, channel unbalancing and uncooperative behavior of nodes.

In this work CLoTH was introduced, a simulator for HTLC payment networks. The CLoTH simulator is a discrete-event simulator which simulates payments on HTLC payment networks and produces performance measures such as probability of payment failure and mean payment complete time. CLoTH is a precise and complete mapping of LN code functions that implement an HTLC payment network: this constitutes the originality of the simulator and guarantees the validity of simulation results. The CLoTH simulator allows systematic analyses of HTLC payment networks.

This work aimed to analyze capabilities and limitations of payment channel networks using CLoTH. Three groups of simulations were discussed: simulations on a snapshot of the LN mainnet dating back to June 2018, in order to find possible non-operative cases, i.e., cases in which a payment is more likely to fail than to succeed; simulations on synthetic networks generated by the simulator, to study the effect of the simulator input parameters on payment network performance; simulations on a snapshot of the LN mainnet dating back to February 2019, whose goal was to analyze the impact of network and protocol modifications on performance.

The first group of simulations has shown that the non-operative cases of the LN mainnet are manly caused by insufficient channel capacities and by channel unbalancing. In fact, when the majority of the simulated payments was between 1 and 10^4 satoshis and the remaining ones (around 15%) were between 10^5 and 10^7 satoshis, around 46% of payments failed because a route with sufficient channel

capacities was not found and around 7% failed because of channel unbalancing. In addition, when the payment rate was set to 100 payments per second, around 25% of payments failed because of insufficient capacities and around 31% failed because of channel unbalancing.

The investigation of synthetic networks has proved that the analyzed networks are characterized by a good performance: the probability of payment success resulted close to 99% in almost all the simulated configurations. Other important findings of these simulations are that in a totally decentralized network without hubs, each node should have a least five open channels to guarantee a well connected network, and that uncooperativeness of nodes does not represent a serious issue in the synthetic networks simulated.

Simulations on protocol modifications have found that the passive rebalancing approach based on fee adjustment reduces by one fourth the probability of failures for unbalancing. Simulations on network modifications revealed that the LN mainnet was resilient to the removal of six hubs, as the probability of payment success decreased only by 0.2% when all the hubs are removed. Moreover, they have shown that, in a typical case of use of the LN in which most of the payments were directed to a few service providers node, up to 26% of payments failed because channel directing to the service providers become unbalanced.

This work, therefore, systematically individuated some of the strong points and limitations of the Lightning Network at its current stage of development. With regard to the strong points, the Lightning Network is resilient to the removal of network hubs and tolerates a contained level of node uncooperativeness. On the other hand, the main limitations of the Lightning Network concern the limited channel capacities and channel unbalancing. Channel capacities strictly limit the amounts of payments that can be exchanged in the network. Also in the most recent snapshot of the network analyzed (dating back to February 2019), when around 60% of simulated payments were between 1 and 10^3 satoshis and the remaining ones between 10^4 and 10^5 satoshis, probability of payment success resulted below 90%, and the main reason of payment failures was the impossibility of finding a route with sufficient channel capacities to transfer the payments. Channel unbalancing causes payment delays - as payments need to be re-attempted when they bump into an unbalanced channel - and payment failures, especially noticeable in the service-provider scenario. The passive rebalancing approach proposed in this work represents a viable solution to address the issue of unbalancing.

The most important contribution of this work is CLoTH, a valuable tool that supports the development of payment channel networks by systematically analyzing their issues, the viable solutions and their evolution. In fact, the investigations presented in this thesis are only the first of a long series that can be conducted using the CLoTH simulator.

In this work, the simulator was effectively used for analyzing protocol modifications concerning rebalancing. In future work, further protocol modifications will

be studied. For instance, to reduce payment failures caused by insufficient channel capacities, the approaches that split a large payment into small ones will be implemented in CLoTH and examined.

CLoTH is also able to simulate attack scenarios in the Lightning Network. In future work, attacks executed by uncooperative nodes will be studied, specifically by nodes that uncooperatively behave after establishing an HTLC. Such nodes, in fact, may cause large delays of payments and lock of funds in the network.

Finally, some improvements will be done to the simulator itself. Currently, CLoTH does not simulate the blockchain underlying the payment channel network. In future work, the blockchain will be introduced in the simulator, to study its interactions with the payment network (e.g., the opening and closure of payment channels). Moreover, an analysis of the simulator execution made possible to identify in the repeated local and sequential application of the Dijkstra's algorithm the performance bottleneck of the simulator. In future work, this limitation will be addressed by making the simulator completely multi-thread and by distributing the computation on a cluster of machines. Simulator performance improvement will permit to conduct additional simulations, specifically on large synthetic networks, to experimentally study and steer the evolution of payment channel networks.

Appendix A

Reference Code Functions

This Chapter describes the details of the `lnd` code taken as reference to develop CLoTH. In `lnd`, there are two main code structures which manage payments: `Switch` and `Link`. `Switch` is the messaging bus of HTLC messages: it is in charge of forwarding HTLCs or redirecting HTLCs initiated by the local node to the proper functions. `Link` is the service which drives a channel commitment update procedure according to the HTLCs that concern the channel. In the following, the functions included in the call graph in Figure 3.1 are described.

- `SendPayment`. As shown by the numbers in the arrows, which represent the order of function calls, this function first tries to find possible routes to transfer the payment to the receiver and then tries to send the payment through one of the routes found. If the payment fails, it re-attempts the payment through another viable route.
- `RequestRoute`. It attempts to find candidate routes which can route the payment to the receiver.
- `findPath`. It runs the Dijkstra's algorithm, using timelock and fee as distance metric. In fact, each channel endpoint has a *policy* which defines the timelock and the fee that will be applied to any HTLC forwarded by that endpoint. The higher the timelock and the fees in the endpoint policy, the higher the distance.
- `newRoute`. It attempts to transform a path into a *route*. A route is a path which connects sender and receiver and which can also transfer the payment. A path is considered capable of transferring a payment if all channels in the path have a capacity greater than or equal to the payment amount, considering also fees. Fee is the amount of funds a channel endpoint withholds as a reward for forwarding a payment through that channel.

- `handleLocalDispatch`. Function of the `Switch` that processes HTLCs relative to payments initiated by the local node. This function returns an error if there are no channels to the next hop with enough balance to forward the payment.
- `handleLocalResponse`. Function of the `Switch` the processes the result of a payment initiated by the local node. It receives an `HTLCFail` or `HTLCFulfill` message and it propagates them back to `SendPayment`.
- `handlePacketForward`. Function of the `Switch` that processes HTLCs of payments initiated by other nodes and to be forwarded by the local node. It produces an `HTLCFail` if no channel to the next route hop has enough balance to forward the payment, or if the local node policy is not respected. The `HTLCFail` message is then sent back to the payment sender.
- `handleDownStreamPkt`. Function of the `Link` which processes HTLCs coming from the `Switch`. It produces an `HTLCFail` if the channel does not have enough balance to forward the payment.
- `handleUpStremMsg`. Function of the `Link`, the first called by a node upon the reception of an HTLC message by another node.
- `SendMessage`. Function to send an HTLC message from a node to another over the network.
- `processRemoteAdds`. It processes an `HTLCAdd` and checks for possible errors. When called by the payment receiver, the function decides whether to accept or not the payment.
- `processRemoteSettleFails`. It processes an `HTLCFail` or `HTLCFulfill` and checks for possible errors.

Appendix B

Complete Simulation Results

B.1 Simulations on the Lightning Network Mainnet

Table B.1 shows the complete results of the first group of simulations on the LN mainnet, discussed in Chapter 4. In each entry of the table, all the independent variables and the output performance measures of the simulations are shown. Only the mean values of the performance measures are presented. The results containing also variances and confidence intervals are available online[1].

Table B.1: Complete simulation results on the LN mainnet (snapshot of June 2018).

r_π (pay/s)	σ_a	F_{sr}	$P_{\bar{c}_b}$	P_s	P_{f_r}	P_{f_b}	$P_{f_{\bar{c}}}$	$P_{\bar{k}}$	T (ms)	L_r (hops)	N_{att}
10	5	0%	0.00%	46.13%	46.11%	7.72%	0.00%	0.03%	388.29	3.17	1.28
10	4	0%	0.00%	50.77%	41.18%	8.00%	0.00%	0.05%	386.36	3.18	1.30
10	3	0%	0.00%	56.87%	33.81%	9.30%	0.00%	0.02%	383.32	3.19	1.31
10	2	0%	0.00%	63.35%	26.63%	9.96%	0.00%	0.06%	373.91	3.16	1.32
10	1	0%	0.00%	65.43%	24.40%	10.11%	0.00%	0.06%	367.83	3.15	1.30
100	1	0%	0.00%	43.88%	25.17%	30.92%	0.00%	0.03%	483.11	3.08	2.30
10	1	50%	0.00%	69.54%	18.81%	11.61%	0.00%	0.04%	467.34	3.36	1.37
10	1	50%	10.00%	57.10%	18.80%	9.21%	14.84%	0.04%	931.53	3.38	1.57
10	1	0%	10.00%	55.21%	24.31%	8.51%	11.92%	0.05%	903.01	3.15	1.51
10	4	0%	10.00%	38.27%	46.10%	6.47%	9.14%	0.02%	902.54	3.16	1.43
10	3	0%	10.00%	47.71%	33.78%	7.66%	10.82%	0.03%	923.16	3.18	1.51
10	2	0%	10.00%	53.64%	26.56%	7.85%	11.91%	0.05%	862.70	3.16	1.51
10	1	0%	10.00%	55.22%	24.30%	8.50%	11.93%	0.05%	902.83	3.15	1.51
10	4	0%	1.00%	49.84%	41.17%	7.95%	0.99%	0.05%	443.00	3.18	1.32
10	4	0%	0.10%	50.79%	41.15%	7.98%	0.06%	0.02%	391.34	3.18	1.29

[1] https://researchdata.nexacenter.org/payment-network-simulator/LN-mainnet-results.zip

B.2 Simulations on Synthetic Networks

Tables B.2 to B.10 show the complete results of the simulations on synthetic networks discussed in Chapter 5. Each Table shows the output performance measures obtained when varying a specific independent variable. Only the means of the output measures are presented. The results containing also variances and confidence intervals are available online[2].

Table B.2: Complete simulation results varying number of channels per node.

N_{ch}	P_s	P_{f_r}	P_{f_b}	$P_{f_{\bar{c}}}$	$P_{\bar{k}}$	T (ms)	L_r (hops)	N_{att}
3	59.61%	23.34%	16.77%	0.08%	0.20%	2052.47	16.80	1.44
5	99.34%	0.31%	0.13%	0.00%	0.21%	1391.92	10.34	1.55
8	99.82%	0.01%	0.00%	0.01%	0.16%	1112.63	8.33	1.51
11	99.86%	0.00%	0.00%	0.01%	0.14%	992.46	7.56	1.47

Table B.3: Complete simulation results varying uncooperative nodes probability.

$P_{\bar{c}_b}$	P_s	P_{f_r}	P_{f_b}	$P_{f_{\bar{c}}}$	$P_{\bar{k}}$	T (ms)	L_r (hops)	N_{att}
0.00%	99.36%	0.31%	0.13%	0.00%	0.19%	1388.51	10.34	1.55
0.01%	99.34%	0.31%	0.13%	0.00%	0.21%	1391.92	10.34	1.55
0.10%	99.27%	0.31%	0.13%	0.10%	0.19%	1432.59	10.34	1.57
1.00%	98.32%	0.32%	0.12%	1.04%	0.20%	1841.17	10.38	1.72
10.00%	87.47%	0.32%	0.08%	11.84%	0.28%	9795.14	10.82	4.55

Table B.4: Complete simulation results varying network topology.

σ_t	P_s	P_{f_r}	P_{f_b}	$P_{f_{\bar{c}}}$	$P_{\bar{k}}$	T (ms)	L_r (hops)	N_{att}
0	99.88%	0.00%	0.06%	0.01%	0.05%	333.16	2.90	1.16
1	99.85%	0.00%	0.07%	0.01%	0.08%	536.83	4.13	1.35
10	99.83%	0.00%	0.07%	0.01%	0.09%	695.73	5.56	1.36
inf	99.34%	0.31%	0.13%	0.00%	0.21%	1391.92	10.34	1.55

[2]https://researchdata.nexacenter.org/payment-network-simulator/synthetics-results.zip

Table B.5: Complete simulation results varying percentage of same-recipient payments.

F_{sr}	P_s	P_{f_r}	P_{f_b}	$P_{f_{\bar{c}}}$	$P_{\bar{k}}$	T (ms)	L_r (hops)	N_{att}
0%	99.34%	0.31%	0.13%	0.00%	0.21%	1391.92	10.34	1.55
10%	99.37%	0.30%	0.13%	0.01%	0.20%	1392.85	10.31	1.56
20%	99.39%	0.29%	0.12%	0.01%	0.20%	1450.04	10.44	1.62
40%	99.43%	0.25%	0.10%	0.01%	0.20%	1473.14	10.37	1.67
50%	83.45%	0.26%	16.06%	0.04%	0.19%	1458.47	10.65	1.88

Table B.6: Complete simulation results varying payment amounts.

σ_a	P_s	P_{f_r}	P_{f_b}	$P_{f_{\bar{c}}}$	$P_{\bar{k}}$	T (ms)	L_r (hops)	N_{att}
1	99.34%	0.31%	0.13%	0.00%	0.21%	1391.92	10.34	1.55
2	99.13%	0.48%	0.19%	0.01%	0.19%	1427.54	10.36	1.63
3	96.80%	2.30%	0.65%	0.03%	0.23%	1611.93	10.40	2.25
4	93.69%	4.67%	1.33%	0.04%	0.26%	1816.37	10.42	2.98
5	91.43%	6.40%	1.85%	0.05%	0.27%	1949.87	10.45	3.47

Table B.7: Complete simulation results varying number of payments.

N_π	P_s	P_{f_r}	P_{f_b}	$P_{f_{\bar{c}}}$	$P_{\bar{k}}$	T (ms)	L_r (hops)	N_{att}
10,000	99.33%	0.21%	0.19%	0.02%	0.25%	1529.99	10.48	1.76
100,000	99.34%	0.31%	0.13%	0.00%	0.21%	1391.92	10.34	1.55
1,000,000	99.41%	0.31%	0.07%	0.01%	0.19%	1273.09	10.22	1.35

Table B.8: Complete simulation results varying number of nodes.

N_n	P_s	P_{f_r}	P_{f_b}	$P_{f_{\bar{c}}}$	$P_{\bar{k}}$	T (ms)	L_r (hops)	N_{att}
100,000	99.34%	0.31%	0.13%	0.00%	0.21%	1391.92	10.34	1.55
200,000	99.35%	0.30%	0.11%	0.01%	0.24%	1559.50	11.05	1.68
500,000	99.16%	0.37%	0.22%	0.01%	0.24%	1777.99	11.96	1.85
700,000	99.10%	0.36%	0.26%	0.01%	0.26%	1858.46	12.29	1.90
1,000,000	99.06%	0.33%	0.33%	0.01%	0.28%	1948.26	12.63	1.96

Table B.9: Complete simulation results varying channel capacity.

C_{ch} (satoshi)	P_s	P_{f_r}	P_{f_b}	$P_{f_{\bar{c}}}$	$P_{\bar{k}}$	T (ms)	L_r (hops)	N_{att}
100	98.37%	0.51%	0.81%	0.03%	0.27%	2134.47	10.17	3.98
1,000	99.36%	0.32%	0.13%	0.01%	0.18%	1404.00	10.33	1.58
10,000	99.35%	0.31%	0.13%	0.01%	0.20%	1392.36	10.34	1.56
100,000	99.34%	0.31%	0.13%	0.00%	0.21%	1391.92	10.34	1.55
1,000,000	99.34%	0.31%	0.13%	0.00%	0.21%	1391.80	10.34	1.55

Table B.10: Complete simulation results varying Gini index.

G	P_s	P_{f_r}	P_{f_b}	$P_{f_{\bar{c}}}$	$P_{\bar{k}}$	T (ms)	L_r (hops)	N_{att}
0.00	99.34%	0.31%	0.13%	0.00%	0.21%	1391.92	10.34	1.55
0.10	99.34%	0.31%	0.13%	0.00%	0.21%	1391.92	10.34	1.55
0.20	99.34%	0.31%	0.13%	0.00%	0.21%	1391.92	10.34	1.55
0.40	99.34%	0.31%	0.13%	0.00%	0.21%	1391.92	10.34	1.55
0.60	99.34%	0.31%	0.13%	0.00%	0.21%	1391.92	10.34	1.55

B.3 Network and Protocol Modifications on the Lightning Network Mainnet

Tables B.11 to B.16 show the complete results of the simulations on network and protocol modifications discussed Chapter 6. Each Table shows the independent variable and the output performance measures ($P_{\bar{k}}$ and $P_{f_{\bar{c}}}$ are omitted since they are always zero). Only the mean values of the performance measures are presented. The results containing also variances and confidence intervals are available online[3].

Table B.12: Complete simulation results on the LN mainnet (snapshot of February 2019).

σ_a	P_s	P_{f_r}	P_{f_b}	$P_{\bar{t}}$	T (ms)	L_r (hops)	N_{att}
1	99.81%	0.19%	0.00%	0.00%	365.35	3.36	1.00
2	99.72%	0.26%	0.03%	0.00%	368.21	3.34	1.01
3	98.09%	1.48%	0.42%	0.01%	377.79	3.35	1.05
4	95.20%	3.63%	1.16%	0.01%	389.82	3.34	1.12
5	92.90%	5.36%	1.72%	0.03%	399.67	3.33	1.18
6	91.35%	6.57%	2.06%	0.02%	407.18	3.34	1.21
7	90.07%	7.46%	2.43%	0.03%	415.68	3.34	1.25
8	89.26%	8.23%	2.48%	0.03%	418.40	3.34	1.27
9	88.85%	8.45%	2.67%	0.03%	422.05	3.34	1.28
10	88.31%	8.94%	2.73%	0.03%	425.98	3.33	1.30

Table B.13: Complete simulation results varying number of hubs removed $N_{\bar{n}}$.

$N_{\bar{n}}$	P_s	P_{f_r}	P_{f_b}	$P_{\bar{t}}$	T (ms)	L_r (hops)	N_{att}
0	95.20%	3.63%	1.16%	0.01%	389.82	3.34	1.12
1	95.29%	3.53%	1.17%	0.01%	399.07	3.44	1.10
2	95.19%	3.64%	1.16%	0.01%	405.77	3.47	1.11
3	94.98%	3.82%	1.18%	0.02%	403.79	3.52	1.11
4	95.02%	3.79%	1.17%	0.02%	422.68	3.58	1.15
5	94.79%	3.93%	1.26%	0.03%	431.99	3.61	1.15
6	94.98%	3.83%	1.17%	0.02%	431.26	3.62	1.15

[3]https://researchdata.nexacenter.org/payment-network-simulator/network-protocol-modification-results.zip

Table B.11: Complete simulation results in the optimal configuration.

(a) Optimal configuration.

r_π (pay/s)	P_s	P_{f_r}	P_{f_b}	$P_{\bar{t}}$	T (ms)	L_r (hops)	N_{att}
10	99.80%	0.20%	0.00%	0.00%	361.98	3.30	1.00
20	99.85%	0.15%	0.00%	0.00%	362.02	3.30	1.00
30	99.85%	0.15%	0.00%	0.00%	361.51	3.30	1.00
40	99.84%	0.16%	0.00%	0.00%	361.24	3.30	1.00
50	99.85%	0.15%	0.00%	0.00%	361.72	3.30	1.00
60	99.85%	0.15%	0.00%	0.00%	361.96	3.30	1.00
70	99.84%	0.16%	0.00%	0.00%	361.75	3.30	1.00
80	99.84%	0.16%	0.00%	0.00%	361.94	3.30	1.00
90	99.84%	0.16%	0.00%	0.00%	362.29	3.30	1.00
100	99.84%	0.16%	0.00%	0.00%	362.40	3.30	1.00

(b) Original LN mainnet.

r_π (pay/s)	P_s	P_{f_r}	P_{f_b}	$P_{\bar{t}}$	T (ms)	L_r (hops)	N_{att}
10	88.89%	9.07%	1.97%	0.07%	403.23	3.33	1.17
20	88.77%	9.09%	2.07%	0.07%	407.25	3.33	1.20
30	88.72%	8.98%	2.24%	0.06%	408.09	3.33	1.21
40	88.65%	8.98%	2.33%	0.05%	411.61	3.33	1.23
50	88.53%	9.00%	2.43%	0.04%	412.96	3.33	1.24
60	88.46%	8.98%	2.52%	0.04%	417.99	3.33	1.26
70	88.45%	8.94%	2.58%	0.04%	418.73	3.33	1.27
80	88.31%	8.99%	2.66%	0.04%	420.71	3.33	1.28
90	88.26%	9.02%	2.69%	0.03%	422.83	3.33	1.29
100	88.31%	8.94%	2.73%	0.03%	425.98	3.33	1.30

Table B.14: Complete simulation results with active rebalancing.

σ_a	P_s	P_{f_r}	P_{f_b}	$P_{\bar{t}}$	T (ms)	L_r (hops)	N_{att}
1	99.81%	0.19%	0.00%	0.00%	365.36	3.36	1.00
2	99.72%	0.26%	0.03%	0.00%	366.84	3.35	1.00
3	98.09%	1.48%	0.42%	0.01%	374.89	3.35	1.04
4	95.21%	3.63%	1.15%	0.01%	384.31	3.34	1.09
5	92.88%	5.36%	1.74%	0.02%	394.81	3.34	1.14
6	91.34%	6.57%	2.07%	0.03%	402.97	3.35	1.18
7	90.08%	7.46%	2.42%	0.03%	408.09	3.35	1.20
8	89.26%	8.23%	2.48%	0.03%	412.69	3.35	1.22

Table B.15: Complete simulation results with passive rebalancing.

σ_a	P_s	P_{f_r}	P_{f_b}	$P_{\bar{t}}$	T (ms)	L_r (hops)	N_{att}
1	99.81%	0.19%	0.00%	0.00%	311.83	2.85	1.00
2	99.72%	0.26%	0.02%	0.00%	313.15	2.86	1.00
3	98.13%	1.52%	0.35%	0.00%	320.75	2.92	1.01
4	95.31%	3.83%	0.86%	0.00%	330.26	2.98	1.02
5	93.04%	5.64%	1.32%	0.00%	330.81	3.00	1.02
6	91.65%	6.80%	1.55%	0.00%	336.62	3.02	1.03
7	90.40%	7.81%	1.79%	0.00%	337.94	3.03	1.03
8	89.63%	8.52%	1.84%	0.00%	338.94	3.03	1.04

Table B.16: Complete simulation results in the service-providers scenario.

σ_a	P_s	P_{f_r}	P_{f_b}	$P_{\overline{t}}$	T (ms)	L_r (hops)	N_{att}
1	99.89%	0.11%	0.00%	0.00%	340.62	3.13	1.00
2	98.78%	0.14%	1.09%	0.00%	350.62	3.13	1.04
3	88.49%	1.13%	10.36%	0.01%	455.52	3.22	1.46
4	78.96%	2.87%	18.07%	0.10%	593.20	3.24	2.02
5	73.56%	4.43%	21.71%	0.30%	687.33	3.26	2.44
6	69.74%	5.60%	24.12%	0.55%	746.51	3.28	2.69
7	66.90%	6.23%	26.17%	0.70%	829.97	3.30	3.03
8	66.30%	6.86%	26.10%	0.74%	867.99	3.30	3.20

Bibliography

[1] Andreas M Antonopoulos. *Mastering Bitcoin: unlocking digital cryptocurrencies*. O'Reilly Media, Inc., 2014.

[2] Albert-László Barabási et al. *Network science*. Cambridge university press, 2016.

[3] Conrad Burchert, Christian Decker, and Roger Wattenhofer. "Scalable funding of Bitcoin micropayment channel networks". In: *Royal Society open science* 5.8 (2018), p. 180089.

[4] David Chaum. "Blind signatures for untraceable payments". In: *Advances in cryptology*. Springer. 1983, pp. 199–203.

[5] Kyle Croman et al. "On scaling decentralized blockchains". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2016, pp. 106–125.

[6] Christian Decker and Roger Wattenhofer. "A fast and scalable payment network with bitcoin duplex micropayment channels". In: *Symposium on Self-Stabilizing Systems*. Springer. 2015, pp. 3–18.

[7] Giovanni Di Stasi et al. "Routing payments on the Lightning Network". In: *Proceedings of the 2018 International Conference on Blockchain*. 2018.

[8] Ittay Eyal and Emin Gün Sirer. "Majority is not enough: Bitcoin mining is vulnerable". In: *Communications of the ACM* 61.7 (2018), pp. 95–102.

[9] Ittay Eyal et al. "Bitcoin-ng: A scalable blockchain protocol". In: *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. 2016, pp. 45–59.

[10] Adem Efe Gencer, Robbert van Renesse, and Emin Gün Sirer. "Service-oriented sharding with aspen". In: *arXiv preprint arXiv:1611.06816* (2016).

[11] Arthur Gervais et al. "Is bitcoin a decentralized currency?" In: *IEEE security & privacy* 12.3 (2014), pp. 54–60.

[12] Arthur Gervais et al. "On the security and performance of proof of work blockchains". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2016, pp. 3–16.

[13] Arthur Gervais et al. "Tampering with the delivery of blocks and transactions in bitcoin". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2015, pp. 692–705.

[14] Yossi Gilad et al. "Algorand: Scaling byzantine agreements for cryptocurrencies". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM. 2017, pp. 51–68.

[15] Mike Hearn. *Anti DoS for tx replacement*. URL: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002417.html (visited on 04/25/2019).

[16] Mike Hearn. *Micro-payment channels implementation now in bitcoinj*. URL: https://bitcointalk.org/index.php?topic=244656.0 (visited on 04/25/2019).

[17] Ethan Heilman et al. "Eclipse attacks on bitcoin's peer-to-peer network". In: *24th USENIX Security Symposium (USENIX Security 15)*. 2015, pp. 129–144.

[18] Jordi Herrera Joancomarti. "Research and challenges on bitcoin anonymity". In: *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance*. Springer, 2014, pp. 3–16.

[19] Raj Jain. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons, 1990.

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

[21] Aggelos Kiayias et al. "Ouroboros: A provably secure proof-of-stake blockchain protocol". In: *Annual International Cryptology Conference*. Springer. 2017, pp. 357–388.

[22] Philip Koshy, Diana Koshy, and Patrick McDaniel. "An analysis of anonymity in bitcoin using p2p network traffic". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2014, pp. 469–485.

[23] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine generals problem". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4.3 (1982), pp. 382–401.

[24] *Lightning Network Specifications*. URL: https://github.com/lightningnetwork/lightning-rfc (visited on 07/31/2018).

[25] Loi Luu et al. "A secure sharding protocol for open blockchains". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2016, pp. 17–30.

[26] Giulio Malavolta et al. "SilentWhispers: Enforcing Security and Privacy in Decentralized Credit Networks." In: *IACR Cryptology ePrint Archive* 2016 (2016), p. 1054.

[27] *Maximum transaction rate*. URL: https://en.bitcoin.it/wiki/Maximum_transaction_rate (visited on 07/31/2018).

[28] Andrew Miller et al. "Sprites: Payment Channels that Go Faster than Lightning." In: *CoRR* abs/1702.05812 (2017).

[29] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2008.

[30] Arvind Narayanan et al. *Bitcoin and Cryptocurrency Technologies: A comprehensive introduction*. Princeton University Press, 2016.

[31] Olaoluwa Osuntokun. *AMP: Atomic Multi-Path Payments over Lightning*. URL: https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html (visited on 07/31/2018).

[32] Dmytro Piatkivskyi and Mariusz Nowostawski. "Split Payments in Payment Networks". In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2018, pp. 67–75.

[33] Joseph Poon and Thaddeus Dryja. *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*. 2016.

[34] Pavel Prihodko et al. *Flare: An Approach to Routing in Lightning Network*. 2016.

[35] *Raiden Network*. URL: https://raiden.network/ (visited on 07/31/2018).

[36] Diane Reynolds. *Simulating a Decentralized Lightning Network with 10 Million Users*. URL: https://hackernoon.com/simulating-a-decentralized-lightning-network-with-10-million-users-9a8b5930fa7a (visited on 10/11/2017).

[37] Adrian van Schie. *Routing Scalable Bitcoin Payments*. 2015.

[38] Yonatan Sompolinsky and Aviv Zohar. *Accelerating bitcoin's transaction processing. Fast Money Grows on Trees, Not Chains*. 2013.

[39] Manny Trillo. *Stress Test Prepares VisaNet for the Most Wonderful Time of the Year*. URL: https://www.visa.com/blogarchives/us/2013/10/10/stress-test-prepares-visanet-for-the-most-wonderful-time-of-the-year/index.html (visited on 04/25/2019).

[40] Bryan Vu. *Exploring Lightning Network Routing*. URL: https://blog.lightning.engineering/posts/2018/05/30/routing.html (visited on 07/31/2018).

[41] Ruozhou Yu et al. "CoinExpress: A Fast Payment Routing Mechanism in Blockchain-based Payment Channel Networks". In: *2018 27th International Conference on Computer Communication and Networks (ICCCN)*. IEEE. 2018, pp. 1–9.

This Ph.D. thesis has been typeset by means of the TEX-system facilities. The typesetting engine was pdfLATEX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete TEX-system installation.