NTNU
Norwegian University of
Science and Technology

An analysis of on-chainLightning Network transactionsin the Bitcoin blockchain

Jardar Tøn

Master in Information Security
Submission date: June 2018
Supervisor: Mariusz Nowostawski, IDI

Norwegian University of Science and Technology
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

An analysis of on-chain Lightning Network transactions in the Bitcoin blockchain

Jardar Tøn

01-06-2018

Master's Thesis
Master of Science in Information Security
30 ECTS
Department of Information Security and Communication Technology
Norwegian University of Science and Technology,

Supervisor: Assoc. Prof. Mariusz Nowostawski

Preface

This paper is a Master's thesis which was carried out in the autumn semester of 2018 in AIMT at NTNU. The main topics of the thesis is Bitcoin and the Lightning Network, which both are two large systems under active development. Both require a background in computer science for getting a detailed understanding of their operation, but the required background information on both will be provided. This project started as a exploration in the interaction between these systems, which resulted in a approach where the Lightning Network is explored trough the Bitcoin blockchian. While this provided a good scope for the project, the project still had exploratory nature until the first results was produced. Even with the bitcoin blockchain as a main approach, implementations for both systems had to be used, and software developed for data collection and processing in relation to both.

01-06-2018

Acknowledgment

I wish first to express my sincere gratitude to my supervisor Mariusz Nowostawski, for his very valuable guidance and support throughout this process.

I would also like to thank Dmytro Piatkivskyi for his help during the planning and initial stages of this thesis.

A special thanks to my classmates for the time well spent together.

Finally, I wish to thank my parents for their support throughout my study.

J.T.

Abstract

The blockchain is one of the main mechanisms enabling Bitcoin to be a decentralized electronic payment system. It provides the system with a shared transaction history, which can be verified by every participant. With increased use, it has become apparent that there is limited possibility for scaling this to handle more transactions. Payment channel networks are one proposed solution to this problem. They allow for more transactions to be done by moving some transactions to a separate network. The Lightning Network is one such network, which uses Bitcoin and the blockchain to operate. While the transactions in this network will not be included in the blockchain, there will be data there related to the network. This is because it needs the blockchain to manage the payment channels which the network consists of.

In this project we have explored the blockchain with the goal of identifying transactions related to the Lightning Network, and by doing so, determine what information about it is available in the blockchain. We have created different methods for identifying these transactions. The methods use different transaction characteristics differing in uniqueness, making some methods more precise, but having fewer results, and vice versa. We created software implementing the methods, which were used to parse the blockchain. The effectiveness of these methods have been quantified by comparing the data we found when parsing the blockchain, to data we collected directly from the Lightning Network. The results shows that the methods are viable for identifying a subset of transactions, and that precision can be sacrificed for finding more. By identifying these transactions we were able to determine what information about the Lightning Network we can see from the blockchain perspective, and also some aspects where we are limited.

We have also adapted heuristics from previous work doing blockchain analysis to our scenario. These were used to link related information we had found when parsing the blockchain, which enabled us to create network graphs showing the relations between the Lightning Network channels identified on the blockchain. While the relations in this network graph were limited, compared to the actual relations found within the lightning network, they show how the blockchain can be used to infer non-explicit information about the lightning network. We have also identified several methods for potentially inferring or locating more information using what is available in the blockchain.

Contents

List of Figures

List of Tables

1 Introduction

Bitcoin is a peer-to-peer electronic payment system relying on cryptography and a distributed ledger known as the blockchain [2]. The purpose of the blockchain is to record all transactions done in the system, and is maintained collectively by the network. New transactions is added to the blockchain by spending computational power; making the transactions practically irreversible as the blockchain grows and more computational power is used, thus removing the need for trusted third parties to verify transactions. Such a decentralized payment solution does however not scale very well. It can handle maximally around 7 transaction per second [1], compared to a centralized payment system like visa, which can handle up to 47,000 transactions per second [3]. A proposed solution to help scale the Bitcoin system is payment channel networks. Such a network allows users to do transactions without them being published in the Bitcoin network, and they are therefore not included in the blockchain, which is why they are known as off-chain transactions. This makes it possible for more transactions being done without increasing the capacity of the Bitcoin system itself. It can be thought of as a layer on top of Bitcoin, since it requires the Bitcoin system and builds on top of it. As the name implies, these networks consists of interconnected payment channels. A payment channel is a one-to-one channel allowing two participants to exchange founds between each other. A network is created by connecting multiple channels with different users, making it possible for transfers to be done across multiple channels-e.g., Alice can pay Bob without having a channel directly connecting them, but using intermediary channels. These off-chain transactions done inside this network does not impact the Bitcoin network, only the management of the payment channels does.

As the blockchain is a public authoritative record of transactions done within the Bitcoin system, it has been used in research projects focusing on anonymity and privacy of users using the system [4, 5]. Because the data in the blockchain provide a complete record of how founds have moved, the focus of the research projects has been to analyze the data on the blockchain, and determine user involvement in transactions and founds. This is done by grouping and contextualizing the information there, and successfully doing so will provide one with a record of all user activity. The Lightning Network (LN) as described in the paper by Poon and Dryja [1], is a payment channel network currently used with Bitcoin. The use of this second layer network allowing transactions to be done off-chain, makes this type of blockchain analysis more difficult. As the Lightning Network (LN) is built on top of Bitcoin, there is a need for on-chain transactions to manage the payment channels it consists of. These on-chain transactions from the LN will be recorded on the blockchain just as other Bitcoin transactions, and the data contained in them will be validated same as any other transactions; they will however not provide explicit record of how founds have moved inside

the LN, unlike the Bitcoin transactions do for the Bitcoin network. This means one can still analyze the blockchain to see how founds move inside the Bitcoin network, but one can not do this to the same degree for the LN. Jordi Herrera-Joancomartí and Cristina Pérez-Solà [6] points this out by stating that the methods used for blockchain analysis in earlier research, is no longer effective if payment channel networks would be widely used. In the case of the LN being widely used, the majority of transactions would be done off-chain and the only on-chain transactions would be the ones required for the payment channels to operate. Giulio Malavolta et al. [7] identifies some possible privacy implications of the on-chain transactions from payment channels networks, so there might be possibilities of using blockchain data related to payment channels networks to reveal user information. Now that there are functioning payment channel networks such as the Lightning Network, we can explore what information will be available from the on-chain transactions; not just in a theoretical manner, but also the impact of real user behaviour and large data sets. In this thesis we will explore the possibility of blockchain analysis in relation to the lightning network, and to what degree it can impact privacy of the users. Doing this while the technology is still in a developmental stage and not yet widely used can be important, as resolving potential privacy risks early is desirable.

This project will explore the implications of LN from the blockchain perspective. We will attempt to answer our research question which is: what information from the LN is available in the Bitcoin system, and how does it impact the privacy of the users? Our main goal, which will allow us to answer this, is to create a method for identifying transactions related to the lightning network in the blockchain. By achieving this we will discover what information is available to us, and we can then determine its use potential use-cases in relation to privacy. We will also explore the relevance of methods used in previous research projects, and how they could potentially be adapted for this new task. Additionally we will identify relevant privacy notions for information in this domain.

2 Background and Related Work

2.1 Background

This section covers the background required for Bitcoin and the Lightning Network. Both systems are large and rely on many mechanisms to operate, so we will only cover the main elements and those relevant to this thesis.

2.1.1 Transactions

Transactions are the mechanism of transferring value between users in the Bitcoin system, but they are also the value themselves. The coins in Bitcoin are more of an abstraction, which we can see clearly from the definition of a coin in the white-paper: "We define a coin as a chain of digital signatures" [2]. Transactions are basically such a chain of digital signatures, so owning "coins" really means controlling transactions, which have a value. Public key cryptography is used to sign transactions and to transfer ownership of value; meaning a transaction is tied to or owned by a public-private key pair. To transfer control of a transaction and therefore its value to another entity, the owner locks the transaction using the public key of the recipient, and then uses their own private key to generate a signature showing that the transfer was done by the now previous owner of the transaction. The cryptographic lock placed on the transaction by the previous owner, makes the recipient in control of the transaction, as they control the private key paired with the public one used to lock the transaction. Everyone else can verify this transfer, because the previous owner generated the signature using their private key when locking the transaction to a new public key, and therefore showing that they had control of the key pair controlling the transaction at the time of the transfer. The transaction as a data-structure, represents one transfer of value, meaning for each transfer a new transaction is created. This results in a series of transactions, each representing a change in control of the value, and these will be related to each other as they contain the same value, but represents different transfers. This is the chain of signatures (transactions) we introduced at the start of this section. By following the chain of transactions and verifying the cryptographic signatures in each transaction, everyone can verify the current ownership of the value. Only the newest transaction in the chain can transfer the value to another key pair, as previous transactions in the chain are only historical states, allowing for the current state to be verified. This transaction chain have the ability to fork and merge, enabling the value to be split into different transactions, or merge the value of many into a single transaction. Therefore the structure is not a linear chain, but a directed acyclic graph (DAG), as no transactions can be spent twice. With the possibility of splitting the value in a transaction, it no longer makes any sense for a single key pair to control the entire value of a transaction. Instead we should define value ownership in terms of controlling transaction outputs, which together with transaction inputs are the two main parts of a transactions.

In short: outputs is the link forward in the transaction chain, while the input is the link backwards. The output of a transaction is where the creator of the transaction locks the value to a public key, and with multiple outputs can give different keys control of different outputs, each with a portion of the value of the transaction. The input of transactions, is the output of previous transactions in the chain, indicating which value the transaction transfers. Figure 1 depicts a small part of a transaction graph with outputs and inputs connecting transactions and transferring control of value. The transaction graph contains all information about how value has been transferred, and as mentioned earlier: only the newest transactions shows current ownership of value, so the leaf nodes in the graph represents the current owners of the value. This graph structure means that the value of a transaction must be used in its entirety when transferring it, or it will be lost, as its transaction is no longer a leaf node. If the creator of a transaction does not wish to transfer the entire value to another entity, they can transfer it back to themselves. This is done by creating a output and locking it to their own public key. Similarly, if a user has no output with sufficiently large value wanted for a transfer, they can use multiple outputs for inputs to the transaction, and by doing so will reach the desired value of the transaction, which an example of can be seen in the right upper transaction in Figure 1.

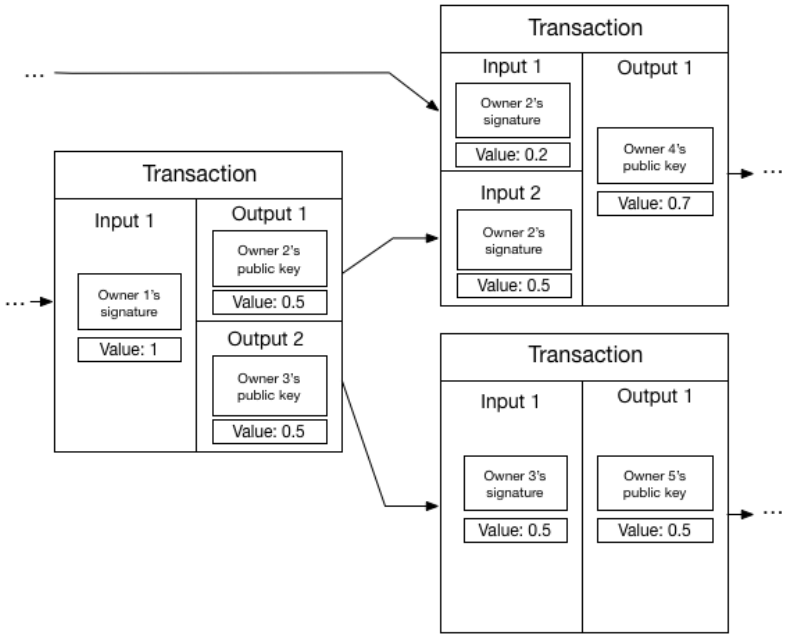

Figure 1: Section of a transaction graph

2.1.2 Blockchain

The double spending problem is that a user can spend the same value twice; in this case create two transactions transferring the value to different users, each transaction spending the same out-

put, and both will be valid as the sender signed both with their private key. To avoid this problem one can have a central authority approving transactions, or in a peer-to-peer system like Bitcoin, create consensus by maintaining a public data-structure containing all transactions. The solution Nakamoto [2] proposed for creating a shared public transaction history without a central authority, is having nodes timestamp collections of transactions by hashing them together with the previous timestamp hash. This is necessary as it makes everyone agree on the state of the transaction history at a given time, which again is used as the basis for the next agreement. If there was no such consensus mechanism, several different transaction histories could be spread within the network and nodes could not determine which one to trust. The structure containing the timestamped collection of transactions is known as blocks, which when created is spread in the network where each node stores a copy. This process of timestamping blocks forms a chain because the inclusion of the hash form the previous block, which also makes the current consensus build upon the previous agreements of the transactions history. This chain of blocks containing timestamped transactions which provides a incrementally created shared history can be seen in Figure 2. When a transaction is included in a block, it is proven to have been verified by the network and is confirmed to have taken place, since it is part of the shared history, and it can therefore be accepted as a payment. If someone wants to double spend an output they would not be able to, because only one of the "double spending transactions" would be included in a block. Reason for this is that the transaction graph would not allow spending the same output in two different transactions, and a single shared transactions graph is ensured with the blockchain, so there would not be two versions of this graph each using a different transaction for the same output. The transaction graph is used to transfer control of value between key pairs, and ensuring that this flow of value is valid; the blockchian ensures that there is consensus in the network about the history of this transaction graph.

As new transactions are created and propagated trough the network, each one node collects them in a pool containing transactions that have not yet been included in a block. Nodes try to create blocks with transactions from this pool by finding a nonce to include in the block, such that the hash of the block starts with a given number of zero bits. This process is called mining; in addition to awarding new Bitcoin to whoever finds the solution first, it also requires computational power to be expended for the hashing. This is called proof-of-work because the solution can be easily verifiable by checking if the nonce produces the given hash, but to find the solution in the first place requires substantial work. As the blockchain grows with more blocks, the total amount of computational resources used also increases. This secures the blockchain as it increases the practical immutability of blocks as they get further back from the tip, meaning they will have a greater amount of spent resources on top of them in the chain. The Bitcoin system is a distributed one, so the mining process is the method used to by nodes to create their shared transaction history. To rewrite parts of the blockchain and change the transaction history, an attacker would have to redo the work from that point and exceed the height of the current used blockchain. This is because the highest chain is considered the valid one as it has the more work put into it by the nodes. Assuming the majority of the nodes in the system is honest and works towards the "correct" chain,

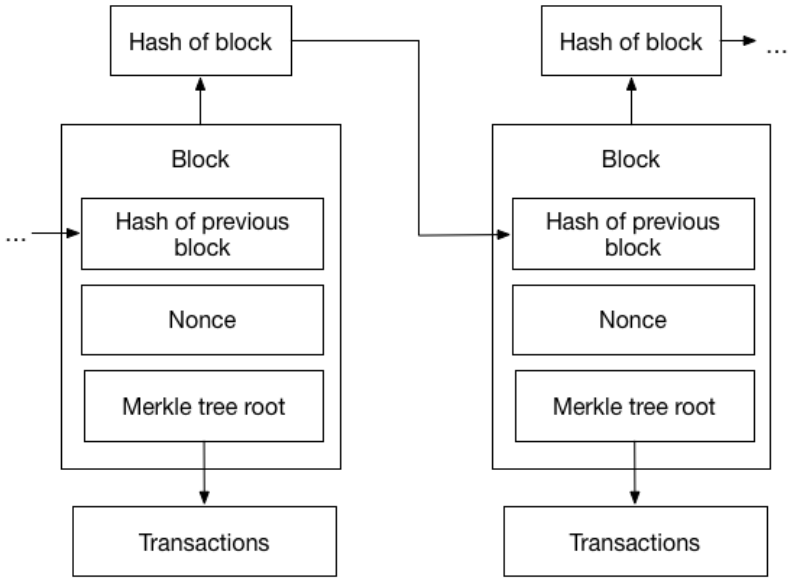

Figure 2: The structure of the blockchain

the scenario to create another chain exceeding the current becomes unlikely and very expensive in terms of work required. There is also incentive of new coins when creating a a new block, so spending computational power to change history in the blockchain instead of mining coins for the current one would be wasteful.

2.1.3 Transaction Scripts

Bitcoin transactions use a stateless, stack based, non touring complete language called script [8]. The outputs of transactions are locked using scripts, which place requirements for spending the outputs; inputs contain scripts that meet the requirements placed by their respective output scripts. The language has a predefined set of operations that can be done, called Opcodes, which limits the possibilities of scripts. An example of a common locking script placed in outputs are:

```
OP_DUP OP_HASH160 <receivers public key hash> OP_EQUALVERIFY OP_CHECKSIG
```

This script is a pay to public key hash (`P2PKH`) script because the sender can use this script to lock the output to the receivers public key hash. The receiver can spend this output by supplying their `<signature> <public key>` in the input, which will meet the requirements placed by the output script and unlock the founds. The unlocking script will be combined with the locking script and executed to ensure that it is valid.

```
<receivers signature> <receivers public key> OP_DUP OP_HASH160
        <receivers public key hash> OP_EQUALVERIFY OP_CHECKSIG
```

The goal of this script is to check if the <public key> in the unlocking part hashes to the <public key hash> of the locking part, and then if the <signature> in the unlocking part matches the public key. When the script executes, the execution pointer will move left to right pushing data to the stack and do the operations specified. In the case of P2PKH scripts, the <signature> will be moved to the stack followed by the <public key>, OP_DUP duplicates the top item on the stack, in this case the <public key>. Then the OP_HASH160 will hash the top item on the stack, which is the top <public key>. The <public key hash> given in the locking part of the script will then be pushed to the stack and OP_EQUALVERIFY will compare the two top items, which now are the <public key hash> and the the hash of the <public key> created by the OP_HASH operation previously, if they are equal they are both removed from the stack. The final operation OP_CHECKSIG takes the two last items on the stack: <public key> and <signature> and checks if the signature matches the public key, which means it was created with the corresponding private key.

There are two "standard" transaction types commonly used: one being the pay to public key hash (P2PKH) already described, and the other is pay to script hash (P2SH) [9]. While users are free to create any locking script they wish within the limits of the language, these types are recognized by most wallet software and allow for senders to transfer value by locking it in known ways. With P2PKH the receiver can just send a hash of their public key, and the sender can create the locking script as outlined above and just insert the hash. P2SH transactions allows for the same process, but using the hash of a script instead of the hash of a public key. This allows for complex scripts to be created, hashed, and the hash value is then used for creating the locking script, resulting in:

```
HASH160 <script hash> EQUAL
```

The unlocking script will contain the script that was used to create the hash as one stack item only. When this is executed the whole original script, known as the redeem script, will be pushed to the stack, and then hashed using OP_HASH. Then the script hash from the unlocking script is pushed to the stack, and finally the two hashes are compared. The unlocking and locking script for this is shown below:

```
<redeem script> HASH160 <script hash> EQUAL
```

This hashing and comparison will ensure that the script supplied is the original one. After it is confirmed in the manner explained above, the redeem script itself is executed to see if it evaluates to true. The conditions for the redeem script must also be satisfied meaning that potential signatures or other data must also be supplied in the input unlocking a output. We can therefore say that that P2SH has two executions, one being to check if the redeem script given by the redeemer matches the hash in locking script as previously explained, and then the redeem script itself is executed. One reason for this is to move the responsibility for supplying these script from sender to receiver. The receiver can create whatever script they wish without the need for the sender to recreate the script when forming the transaction and locking the output. The sender receives a hash of a script and

can crate the standard P2SH locking script with the format given above. The receiver needs to know the script to unlock the output, but the sender does not need to know the script to transfer value; it therefore makes sense to place the burden of supplying the redeem script on the receiver instead of the sender. A typical use case for P2SH, is multisig redeem scripts:

```
2 <public key 1> <public key 2> 2 OP_CHECKMULTISIG
```

These scripts require multiple signatures and therefore multiple key-pairs to be valid, which implies multiple users are required to spend a output. In the case above two signatures is required for the two keys, indicated by the number 2 before the keys and the 2 after. This will be the redeem script which is hashed and given to the sender for them to create the P2SH locking script with this hash. The sender does not need to know the public keys required to spend the output or that it is a multisig script in the first place. The receiver however, needs to know as they are the ones who will claim the value and will therefore likely control at least one of the public keys required in the redeem script.

As we have explained, public-private key pairs are used when transferring control of the value represented by a transaction output, by creating new transactions and locking their outputs to the public keys (or a script with a key) of the receiver. Bitcoin address is an abstraction of this concept. In the P2PKH transctions discussed above, a hash of the public key is used when creating the locking script; address is a base-58 encoded public key hash which is easier to use than a public key hash directly [8]. Reason for this is that base-58 does not contain letters and numbers which can be easily confused, for example, number zero 0 and capital letter O. It also has a prefix denoting if it is a P2PKH or P2SH address, and a checksum at the end which can detect some common errors in the address.

2.1.4 Segregated Witness

Segregated Witness (segwit) is a change to mainly the Bitcoin consensus layer defined in Bitcoin improvement proposal (bip) 141 and 143 [10, 11]. It changes the structure of the validating data used to verify transactions and where this data is stored. As mentioned, transactions outputs have a locking script and a unlocking script; the witness is the unlocking script, which enables us to check the validity of the output spent by using it with the locking script. Segwit places the witness or unlocking script in a witness data structure which is no longer stored inside the transaction it is used to validate. One of the main reasons this is done is to fix transaction malleability, which is the ability to change the transaction id of a transaction. This id is a hash of all the properties of a transaction including the unlocking script/witness. The only malleable parts of a transaction is the unlocking script, because a transaction is unique and outputs can only be spent once-i.e., if the choice of inputs, choice of outputs, or the value of these changes, it is no longer the same transaction. Therefore, when the transaction has been created most of it is non malleable, but the unlocking script can be modified in different ways without invalidating the transaction. Adding operations to the unlocking scripts or zero padding the numbers used can achieve this effect, and

thus change the id, while transaction is for all intents and purposes the same [12]. With segwit, the transaction hash will be calculated without the witness data, and therefore remove possibilities for malleability. As the witness data is only used to validate the transaction itself, it is not necessary to determine the state of the blockchain and can therefore become optional for lightweight nodes wishing to save bandwidth.

Changes in the consensus mechanism within the Bitcoin network could cause the blockchain to fork. If one set of nodes upgrades their software having other consensus rules, they would only create blocks containing transactions following those rules. Similarly the nodes using the old rules will only use the transactions matching their specifications. The blockchain will therefore split from the point which they last agreed on, resulting in two different directions each containing their separate transaction history. This is known as a hard fork because of the effects it has on the network and the blockchain. Segwit was designed to avoid this by being backwards compatible with non segwit nodes. Such a change is known as a soft fork since it does not cause a hard fork. Soft forks are easier to introduce to a decentralized network because it does not require a coordinated upgrade of software. A segwit transaction ouptut is constructed such that a non segwit enabled node will interpret the output as being spendable by everyone-i.e., reedemable by a empty signature; this means a non segwit node cannot correctly validate a segwit transaction but will not consider it invalid [8].

Segwit has two "standard" transaction types which mirrors the functionality of the existing ones discussed in Section 2.1.3: pay to witness public key hash (`P2WPKH`), and pay to witness script hash (`P2WSH`). The main change is how the locking scripts in the outputs is structured; instead of including the script in the output, there is only a version byte with the value 0 to 16 followed by a 20 bytes containing a public key hash, or 32 bytes containing a script hash. The amount bytes found after the version byte allows us to determine if the output is a `P2WPKH` or `P2WSH`. Currently, only version 0 is used with the rest being reserved for future versions [10]. An example of a `P2WPKH` locking script is:

```
0 <20 byte public key hash>
```

The unlocking script is found in the witness structure which is a stack containing the required data to validate the transaction. In the case of the `P2WPKH` the witness stack will contain two elements: the `<signature>` and the `<public key>` which is the same things required to validate the old `P2PKH` transaction type. In the case of the `P2WSH` type, the witness stack can contain a varying amount of items depending on the requirements of the redeem script. If the redeem script is a mutlisig script the stack will contain the required amount of signatures, with the last item of the stack always being the redeem script itself.

2.1.5 Payment Channels in the Lightning Network

The structure and function of the Payment as channels explained in this section will be as they are described in the lightning network paper by Poon and Dryja [1]. A payment channel allows for two

people to connect and transfer founds between themselves. By many users creating channels with each other, a network of channels is formed, which we will discuss in detail in Section 2.1.6. Here we will focus on how a channel allows for two people to transact in a trustless manner. The payment channels use Bitcoin transactions both for managing the channels, and to send founds inside the channel. But not all of them will be broadcast to the Bitcoin network, and therefore not included in the blockchain. At a high level, the channel is used by the two participants to keep track of how founds is distributed between themselves. This means that the channel has a balance showing how much founds each of the two participants has. The balance is updated as founds is sent between them. This is the basic premise of payment channels: when two people sends founds between themselves, it should not be necessary to broadcast each transfer, instead only the final result. E.g., Alice and Bob have a channel with a value of one coin, with the initial distribution being 0.5 to each; Alice sends Bob 0.1 coins in three separate transactions, resulting in a final distribution of 0.2 to Alice and 0.8 to Bob; this result is broadcast while the three 0.1 transactions are not. Using the example above we can see that only publishing the result requires one transaction to be broadcast, in comparison to three transactions if all intermediary transactions were broadcast. The transactions that are broadcast we refer to as on-chain transactions, since they will be included in the blockchain. Off-chain transactions are the intermediary transactions not broadcast, and therefore not found on the blockchain. While payment channels allow for multiple transactions to be done off-chain, they do require some on-chain transactions for managing the channels. One of these on-chain transactions are the founding transaction, which is used to create the channel and make a common starting point for the users in the channel. The founding transaction will contain a output locked using a multisig script, which we discussed in Section 2.1.3. The multisig redeem script used for the output creating the channel has the form [13]:

```
2 <public key 1> <public key 2> 2 OP_CHECKMULTISIG
```

Multisig script requires multiple signatures from different keys to be spent. The script above is a 2of2 multisig script when we denote it in the form of n of m, where the number n is the required and m is the potential number of keys. Using 2of2 mutlisig scripts for the payment channel output in the founding transaction means that both parties in the channel must sign the transaction to spend it. This collectively controlled output in the founding transaction located on the blockchain, is the starting point of their channel, enabling them to do off-chain transactions. But to avoid the value of the channel being stuck in the founding transaction because one or both parties are uncooperative, as spending it the output on-chain requires both signatures, a new transaction is created with both signatures, spending the output and refunding the value. This is done before the founding transaction is published, such that either party have the ability to close out the channel by publishing the refund transaction. The process of setting up a channel starts with the parties creating a founding transaction with a 2of2 multisig output, but they will not exchange signatures for this transaction before they have created the refund transaction. The exchange of signatures is done in a specific order where first the refund transaction is signed by both sides, and only then are signatures exchanged for the founding transaction, and it is then published to the blockchain. The

channel has now been established in a trustless manner, and it also allows for refunds initiated by either party by publishing the refund transaction and closing the channel.

The off-chain transactions done between the parties inside the channel is known as commitment transactions. They contain the current distribution or balance in the channel, which is how the total value of the channel will be split between the parties. Each time a transfer is done inside the channel a new commitment transaction is created reflecting the new balance between the two parties. E.g., the total value of the channel is 10 coins and the balance is 5 to Alice and 5 to Bob, which is reflected in the most recent commitment transaction; Alice decides to send Bob 1 coin; now the balance is 4 coins to Alice and 6 to Bob and a new commitment transaction is created to reflect this change. This is shown in Figure 3 where we see the first commitment transaction on the left, and the new on the right which reflects the transfer from Alice to Bob used in the example. We can also see that commitment transactions are created from the output of the founding transaction just as the refund transaction was. The balance in the channel is reflected in the outputs of the commitment transactions-i.e., there will a output for each of the parties in the channel with the their portion of the total value. The refund transaction can be said to be the first commitment transaction since it describes the initial balance in the channel. This initial balance will be whatever each party contributed to the input of the founding transaction. When the balance in the channel needs to be updated a new commitment transaction is created, which spends the output of the founding transaction again. While it is not possible to spend output twice, the commitment transactions has not been published, so the output is not really spent. Off-chain transaction consists of creating new commitment transactions for each transfer, reflecting the new balance in the channel.

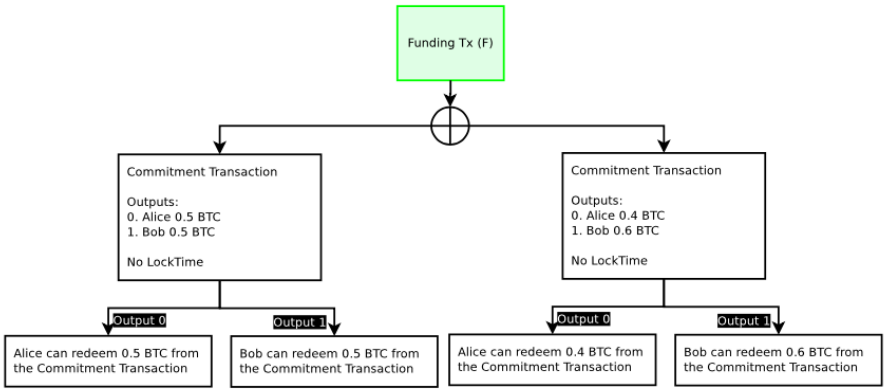

Figure 3: Commitment transactions. Source: [1]

Alice and Bob exchanges signatures for each newly created commitment transaction. This is satisfy the multisignature condition on the founding transaction output used in the channel, allowing any one of them to publish the commitment transaction to the blockchain at any time if they

wish. As a transaction output can only be spent once, so any commitment transaction published to the blockchain will spend the founding transaction, making any other commitment transaction now invalid and unspendable. This will also close the channel, meaning it can no longer can be used, because its founding transaction is spent. Channels are closed out by publishing a transaction spending the output of the founding transaction. Therefore, to be able to do many transfers between the parties in the channel the commitment transactions should ideally not be published immediately. Only when something goes wrong, or both/one of the parties wishes the channel to close, should the newest commitment transaction representing the current balance be published to the blockchain, and thereby closing out the channel and distributing the value. This results in only two on-chain transactions: the founding and closing transaction, with a potential of many off-chain commitment transactions being done in-between.

As the payment channels should be trust-less, there is mechanisms that ensures cooperation-e.g., the refund transaction and founding transaction signing order, which makes locking the founds used in the channel impossible. With the newest commitment transactions reflecting current balance in the channel, older ones will contain a different and therefore wrong balance, and should therefore not be published. Using the earlier example where a channel between Alice and Bob had the value of 10 coins and the balance was 5 to each of them. If Alice Does several payments to Bob and the balance ends up being 0 to Alice 10 to Bob, she could publish the commitment transaction where the balance was 5 to each and get all her money back. To deal with this problem the there is a penalty to anyone who publishes any other commitment transaction than the latest. The penalty is that whoever published the old commitment transaction will loose all the founds to the other person in the channel. As both has the ability to publish a old commitment transaction, we must determine which of the two published; this is described in the Lightning paper as the problem of ascribing blame [1]. When the two parties exchanges signatures for a commitment transaction, we end up getting two different versions of the same transaction. Bob signs one and sends it to Alice, and Alice signs one and sends it to Bob. This means both sides ends up with a half signed transaction which only needs their own signature before it is valid and can be published to the blockchain. The multisignature requirement means that each of them can only publish the one they received from the other. If Bob is the one who published a commitment transaction we can find out, because he must publish one of those he received from Alice containing her signature. As each side receives their version of the commitment transactions from the opposing party, the opposing party can create a insurance clause on the output to the party receiving the commitment. This means that when Bob signs a commitment transaction for sending to Alice, he can place a insurance clause on the output for Alice, punishing her if the commitment is published when it is outdated, and vice versa. This means that the commitment transactions for the two parties is not exactly the same, because they will contain this insurance mechanism, but the transaction pair will still have the same balance, spending the same outputs and have the same outputs, but with the difference of which output this insurance is placed on.

The mechanism to punish the party who publishes an old commitment transaction consists of locking their output for a some period of time, meaning they cannot spend it immediately, while also allowing for the the other party to spend that output. This will result in one party getting all the founds in the channel if the other published an old commitment. The timelock of the transaction outputs is done with Revocable Sequence Maturity Contracts (RSMC). This will be placed on the output to the party who has the ability to publish the commitment transaction. The reason for this is that the publisher would try to steal value by publishing a commitment transactions where the value favoured them compared to the most recent commitment. For each set of commitment transactions each of them will have a timelock/RSMC on the output giving founds to the holder of the commitment transaction. This means that parties can only publish commitment transactions with timelocks on their own outputs. The output which gives founds to the other party is not encumbered with this timelock, since one would not publish a old commitment transaction where the balance favors the other. Figure 4 shows this structure with two sets of commitment transactions and their outputs. The different colors indicate the ability for one party to publish a transaction and spend a output: Alice can broadcast the purple transactions and Bob the blue ones. The timelock is enforced by the number confirmations on the commitment transaction. A transaction is confirmed when included in a block. When more blocks get added on top in the chain the transactions have more confirmations [8]. For the timelock/RSMC case, it will make the party which publishes the commitment transaction unable to spend their output until the transaction has a set number of confirmations. In the Lightning Network paper [1] they uses 1000 confirmations, which will take a while when considering a 10 minute block interval.

The need to wait for confirmations to spend the output is enforced on every set of created commitment transactions. It enables the punishment as it allows to other party to spend the output before the timelock expires. However, the punishment only applies to old commitment transactions, so the punishment is only made possible when a commitment transaction pair is invalidated by creating a new one. The invalidation or revocation of old commitment transactions is done by creating a Breach Remedy transaction which spends the same output as the timelocked/RSMC output. Each of the parties signs this transaction spending the timelocked/RSMC output of their own commitment transaction, and then gives it to the other party to revoke it. This enables both of them to spend the timelocked output belonging to the other party if they should publish a revoked commitment transaction-e.g., Alice creates a breach remedy transaction spending her output in the old commitment transaction and sends it to Bob, and he does the same; if either of the two parties now publishes a old commitment transaction, they cannot spend their output immediately because of the RSMC/timelock, but the other party can publish the breach remedy transaction, and also spend their own output and thus get all founds in the channel. However, spending the timelocked output should to be done before the timelock expires as the party which published the old commitment can now also spend that output. Both parties should therefore observe the blockchain so they can see if an old commitment transaction is published, allowing them to use the breach remedy transaction before the timelock expires.

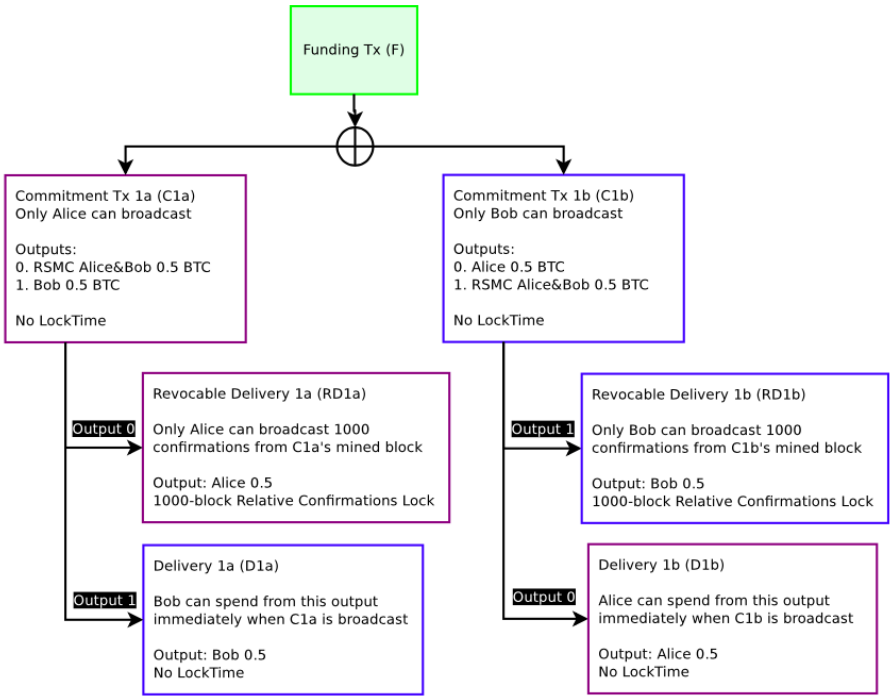

Figure 4: Different commitment pairs. Source: [1]

Here we will present an example outlining the chain of events occurring if a old commitment transaction is published. Alice and Bob has a channel with the value of 1 coin, to which they each contributed 0.5 coins. They create a funding transaction, a refund transaction which will be the first commitment transaction, paying back 0.5 coins to each, they sign it, and then sign the founding transaction. After the funding transaction is published to the blockchain the channel is ready as can be seen in Figure 5. We can also see in the figure that the current commitment transaction pair (C1a for Alice and C1b for Bob) has two outputs giving each 0.5 coins, one normal (D1b and D1b) and one timelocked Revocable delivery (RD1a and RD1b). Alice then wants to send Bob 0.1 coins. To revoke the current commitment pair Bob creates his breach remedy (BR1b) which spends his 0.5 of the commitment transaction (C1b); this is the same 0.5 output that is used in the Revocable delivery (RD1b). Bob's 0.5 output from his commitment transaction (C1b) can now be spent either by waiting for the timelock (RD1b) or Alice can spend it using the breach remedy (BR1b) she received from Bob. Alice does the same thing and sends it to Bob. Now that the old commitment transactions (C1a and C1b) are revoked, they can create the updated pair (C2a and C2b) witch the new balance which outputs 0.4 to Alice and 0.6 to Bob. Now Bob decides to broadcast the old commitment transaction (C1b) to the blockchain. A strange decision since in the old balance he has 0.1 coins less, but it is still a violation of the agreement in the channel of not broadcasting old

commitments. He cannot claim his Output1 with 0.5 coins from the published commitment (C1b) immediately, as he must wait for 1000 confirmations to do so because of the Revocable delivery (RD1b). Alice sees on the blockchain that an old commitment transaction has been published. She has access to her output0 with 0.5 coins from the published commitment (C1b) at once, as it was Bob who published the commitment. She also publishes the Breach remedy transaction (BR1b) she received from Bob when they revoked the commitment transaction. This spends Bob's output and she will therefore get the all the value in the channel. The fig.5 shows this process with green indicating that a transaction is published to the blockchain and therefore invalidating the other transactions. We can also see how the breach remedy transactions (BR1a and BR1b) are able to spend the same outputs as the the timelocked revocable deliveries (RD1a and RD1b).

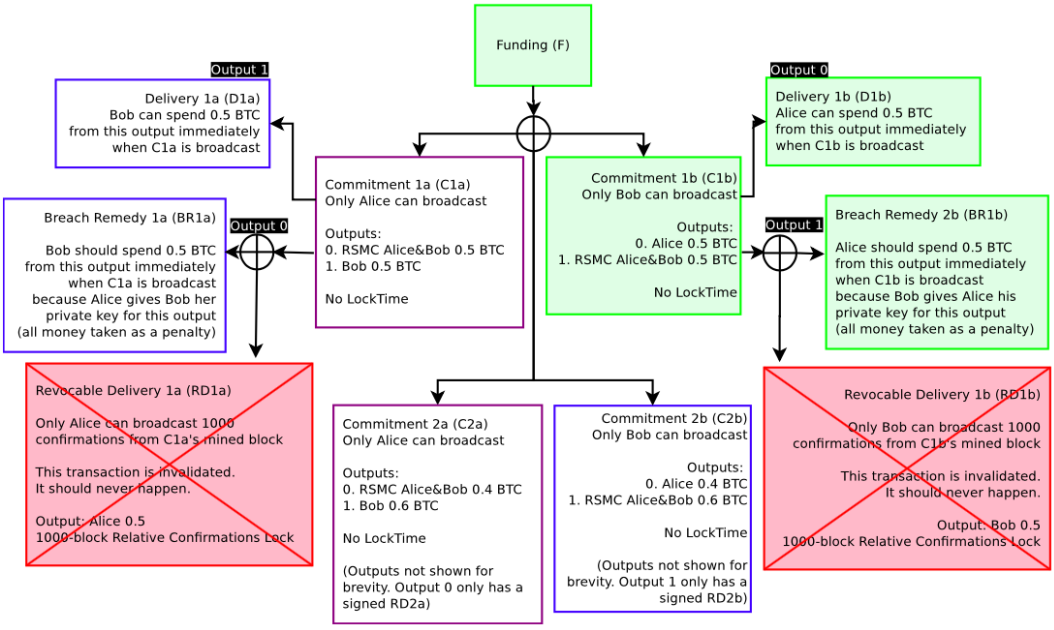

Figure 5: Publishing old commitment transactions. Source: [1]

Publishing a old commitment transaction as described above will spend the founding transaction and close the channel in addition to the other implications already described. If the most recent commitment is published it will also close the channel and place a timelock on the output to the party who published it, but there will has been no Breach Remedy transaction exchanged for that commitment pair, since it has not been revoked, so each party will only be able to spend their own output. The channel can also be cooperatively closed if both parties agrees to do so. They simply create a new transaction spending the founding transaction with the outputs reflecting the balance in the latest commitment. They exchange signatures and it can be published on the blockchain. The outputs of this transactions will not have any timelocks, so each party will be able to claim

their founds when they wish. This is because they both have to agree to close a channel this way, if they cannot agree, one party can simply publish their newest commitment transaction and close the channel that way, but they will need to wait for the timelock. Closing a channel cooperatively means that the total number of transactions published to the blockchain will be less, one founding and one closing, instead of the extra timelocked (or potential breach remedy) transactions.

2.1.6 Networked Payment Channels

The payment channels is used by two entities to exchange founds with each other. The LN is formed by having many such interconnected channels, creating a network. This means that a user has the ability to send founds to others across other users payment channels. If everyone needed to create a payment channel with everyone else it would not be very useful or effective, so there are additional mechanisms that allows for payments to be routed across other users payment channels. As explained in the previous section, the payment channels used in the LN does not require trust between the two parties in the channel. There is also mechanisms that makes sending payments across channels not requiring the sender to trust the intermediary nodes along the path. The construction that enables this is called Hashed Timelock Contracts (HTLC). It creates contracts where intermediary nodes have a guarantee that they can get founds from the sender, if they first transfer it to the receiver. By having the intermediary nodes first pay the receiver and then get the founds from the sender, one stops the intermediary nodes from not passing the founds along to the receiver. If the intermediary was sent the founds first, they could simply not pay the receiver and keep the founds. But for the intermediary nodes to accept this, they need to be sure that the sender will pay them what they have already given the receiver. This is the guarantee provided by the HTLC construction. It uses a hash function with an input or preimage `R`; the receiver inputs `R` into the function and gets a hash `H` which is given to the sender. The sender then promises to pay the intermediary if they can provide `R` that generates `H`. The intermediary then makes the same promise to pay the receiver if they can provide a `R` that generates `H`. Now a chain of promises has been created, each promise being to pay a amount with condition of providing the input `R` which generates `H`. Because it was the receiver who created `H` using `R`, they know `R` and can therefore give it to the intermediary and get the founds from them. `R` has now been disclosed to the intermediary, which in turn can use it to get the founds from the sender as previously promised. This can easily be extended to include more intermediaries.

The promise to pay someone in exchange for disclosing `R` generating `H` is not simply a promise. It is a part of a transactions which needs `R` to be valid. When the sender promises to pay the intermediary, the sender is no longer in control of the founds because a transaction containing this clause has been created and given to the intermediary. So the founds are in a way transferred in a manner and direction one would expect (from sender to intermediary, then intermediary to receiver), but having the clause of providing `R` generating `H` to be claimed. A timelock is used to ensure that if no R is provided, the founds can be returned back to the creator of the transaction-i.e., the one who made the promise to pay if provided the right `R`. This ensures that the founds will either be delivered

or returned by setting a deadline for providing R. If no such thing existed, the series of transfers needed could be delayed or stopped completely by nodes not providing R. Using an example, and both Figure 6 and Figure 7 we can see how this works in detail. Alice wishes to send founds to Dave, but to do so she must first send them through Bob and Carol. Dave the reciver creates the hash H using the preimage R and sends H to Alice. She sees that there are tree hops for the founds to reach Dave, so she creates the first HTLC with a three day timelock, and sends it to Bob shown as step 1 in Figure 6. To re-iterate: the HTLC is part of a transaction where the preimage R has to be given for the founds to be claimed, if R is not given within the timelock timeframe (three days in this case) the founds can be reclaimed by the creator of the HTLC. Bob creates a new HTLC for Carol with the same condition except that the timelock is two days shown as step 2 in Figure 6. In step 3 Carol creates a HTLC for Dave with the timelock set to one day. At this point the founds have been sent but no one has claimed the founds from the HTLC's, so the payment is not complete. If Dave does not disclose R and claim the founds from the HTLC he received it will time out, and so will eventually the other HTLC's, meaning all founds will be returned.

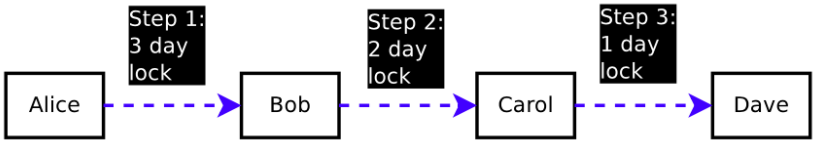

Figure 6: Series of HTLC's between nodes. Source: [1]

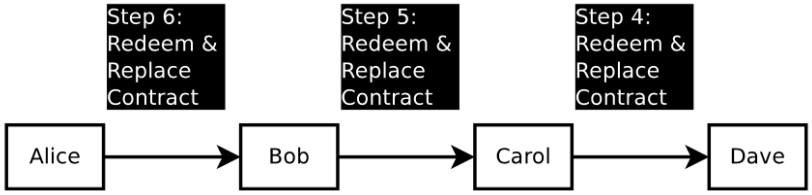

Figure 7: HTLC founds claimed. Source: [1]

In Figure 7 we see how the payment is completed with steps 4, 5 and 6. As it is Dave who knows R, he must start the process. In step 4 he reveals R to Carol to claim the founds in their HTLC. Carol can then do the same with Bob in step 5, and finally step 6 where Bob can claim his founds from Alice. Now the founds have been fully transferred from Alice to Dave. The reason that the timelock is shorter the closer we get to the final destination of the payment, is so it ensures that each person has time to claim the founds from the preceding one. Lets say that step 3 in Figure 6 had a timelock of 5 days, and Dave waited 4 days to disclose R to Carol and get founds from her. The HTLC from Bob to Carol would have expired by then and Bob would have reclaimed his founds, meaning Carol

has transferred founds to Dave without the ability to get founds from Bob.

2.1.7 HTLC inside the payment channel

The previous section explained the high level functioning of HTLC's and how they are used to transfer founds across different payment channels. Here we will explain how the HTLC construct interacts with the already established dynamic within a payment channel. A HTLC inside a payment channel will be a output of a commitment transaction. The outputs of commitments reflects the balance between the parties inside the channel as previously mentioned, so by having another output we can also reflect HTLC founds in transit. The HTLC output is a special type of output; it is a output script which has two different paths to redeeming the output: one is by providing the preimage R, and the other is by waiting for the timelock to expire. As an output of commitment transaction the HTLC will use some of the founds available in the channel, which is determined by the founding transaction. E.g., the value of a channel is 10 coins with a 5 to each balance, a HTLC for 1 coin will be reflected in the commitment transaction as the sender of the HTLC having only 4 coins and the reciver havning 5, but the total value of the channel will still be 10 (4 sender, 1 HTLC, and 5 receiver). If they both agree on who can spend the HTLC output, with either the receiver knowing the preimage R which means the receiver can spend it, or the timelock has expired enabling the sender to spend it, then they can cancel the HTLC by creating a new commitment transaction reflecting the new balance. This new balance is either the receiver getting the value of the HTLC, or the sender getting it back. This is how HTLC's are resolved off-chain when the parties agree on either parties ability to spend the HTLC output, and they therefore create new commitments to reflect this. If they cannot resolve the HTLC in this manner they can do so on-chain, similarly as they could with commitment transaction by broadcasting them if they could not close the channel cooperatively. The two ways to spend a HTLC output on-chain is either by waiting for timelock to expire, or provide the preimage R, both creating a transaction spending the output. As these transactions spend a output from a commitment transaction, the commitment transaction itself would also need to be broadcast on the blockchain, so one be able to spend its outputs. As explained earlier both parties have the opportunity to broadcast their commitment transaction, but restrictions will apply for the publisher. The timelock, and revocation of old commitments with breach remedies, enforces the agreement of not publishing revoked commitments. The same mechanism is used for the HTLC output in the commitment transactions.

Using the example of Alice and Bob having a 1 coin channel with a 0.5 to each balance we can see how this works. Alice sends Bob 0.1 coins using a HTLC, and they create a new pair of commitment transactions (C2a, C2b) as shown in Figure 8, where Alice has the ability broadcast the purple squares ones and Bob the blue. If the sender, in this case Alice, is the one who broadcasts the commitment transaction (C2a), she will have the restrictions discussed previously placed on her. As Alice published the commitment, the left side of Figure 8 will be used, as Bob's commitment (C2b) wil be invalid. The commitment transaction (C2a) published has three outputs: output0 is 0.4 to Alice herself (RD2a), but since it is her commitment transaction it will be timelocked; output 1 is

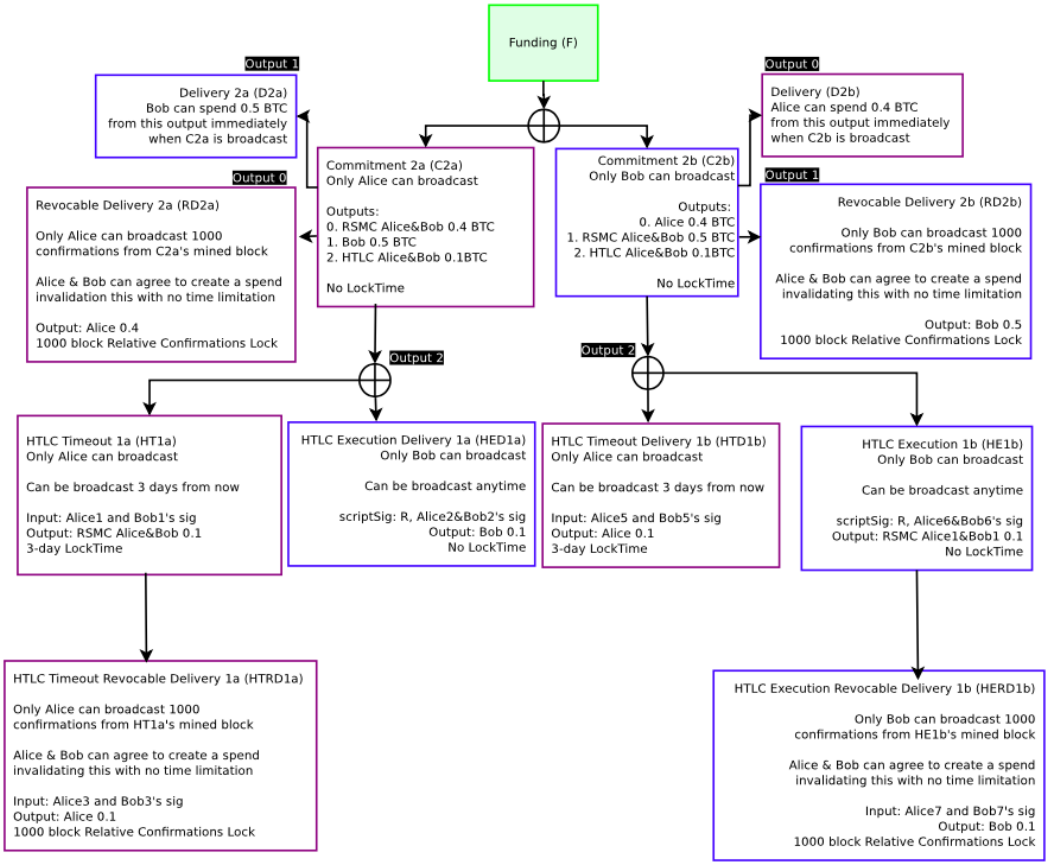

Figure 8: HTLC outputs in commitment transactions. Source: [1]

0.5 for Bob, which he can spend immediately (D2a); output 2 is the HTLC output, which has two different output paths, Bob can spend it at once by creating a transaction containing the preimage `R` (HED1a), or Alice can spend it (HT1a) when the 3 day timelock expires. But because Alice was the one publishing the commitment, her transaction spending the HTLC output will also be timelocked. The timelock of the HTLC output was to ensure that HTLC's are either delivered or refunded. This timelock being placed on the transaction spending the HTLC is to enable the same punishing mechanism as discussed earlier. When old commitment transactions are revoked, so should the HTLC's. This means the parties exchange keys for the transactions spending the HTLC output. So if an old commitment transaction is broadcast, the party who broadcast it will also loose the HTLC output in addition to the normal output, since the other party will be able to spend it and broadcast to the blockchain before the timelock expires. Similar as the commitment transactions, each party will have their own version timelocking their outputs as can be seen in Figure 8.

Similarly to channel operation without HTLC's, it is desirable to avoid publishing to the blockchain unless it is necessary. The parties can cooperate and agree on new commitment transactions with or without HTLC outputs. To resolve a HTLC one of the parties must simply prove to the other that they are capable of spending the HTLC output, and can do so on the blockchain if they wish. If Bob is the receiver he can publish the commitment transaction, and if he has the preimage `R` he can also create a transaction spending the HTLC output. As he is the broadcaster he must wait for confirmations on both, but will eventually receive his founds. If Bob does not have the preimage `R` and the timelock has expired Alice can now spend the HTLC output. Since they can prove to the other party that they have this possibility, they can instead create a new commitment transaction pair with the HTLC value either transferred to Bob or returned to Alice, thereby keeping commitment transactions off-chain and have the channel remain open. When doing this they need to invalidate their old commitment transactions and also the HTLC output of these by exchanging keys allowing the other to spend their transaction spending the HTLC output. This allows the parties to transfer founds with a HTLC inside a channel in a trustless manner, which again enables transfers across multiple channels. No transactions needs to be broadcast to the blockchain unless one party is unresponsive or unwilling to cooperate. But with the addion of HTLC's inside a channel, the total transactions published on-chain will be higher, if they cannot be resolved off-chain. When using the HTLC construct to send founds across multiple channels as shown in Figure 7, broadcasting to the blockchain as discussed in this section will ensure the transfer will finish, in the event of a node not cooperating as illustrated in Figure 9.

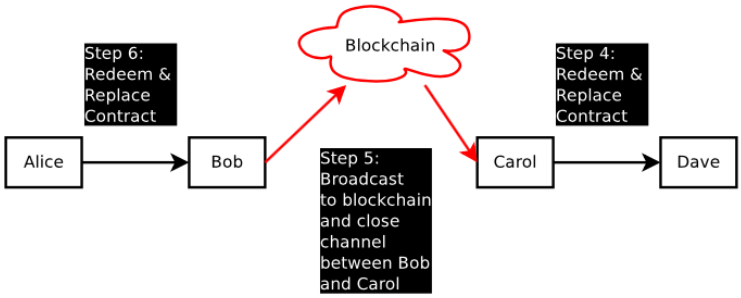

Figure 9: Settlement of a HTLC on the blockchain. Source: [1]

2.2 Related Work

Most of the work already done regarding blockchain analysis does not consider the LN, as the work proceeds the inception of the LN. But their approach and methods can nevertheless be relevant for this project. The fact that the blockchain is a public distributed data-structure containing all transactions done by every Bitcoin user, means that is is by its definition something that limits privacy. Analyzing the blockchain can theoretically give us all records of all transactions related to a single individual; this will include the amount sent and received, the time and date this was done, and

which addresses these transactions was going to or coming from. As discussed in Section 2.1.3 the addresses are encoded public key hashes, and keys can be generated at will, so users are free to use new addresses whenever they want. It is therefore considered best practice to generate a new address for each transaction, because doing so results in transactions between different addresses each time. For the rest of this paper we will refer to keys instead of addresses, since addresses is just hashed and encoded public keys. The practice of not reusing keys means there is no longer a 1-to-1 mapping between users and keys. This use of new keys for each transaction gives the users Pseudonymity, which is defined by by Pfitzmann and Köhntop [14] as: "Pseudonymity is the use of pseudonyms as IDs.". They also define pseudonymity in relation to linkability, which depending on the context has different levels of anonymity. One of these which is very relevant for our scenario is the transaction pseudonym. For this context they explain that for each transaction a different pseudonym is used, like a random number. This makes different transactions pseudonyms hard to link by themselves, which will give it strong anonymity. The transaction pseudonyms is exactly how the Bitcoin system enables some privacy for its users. Having a new pseudonym for each transaction makes it difficult for observers to link the activity of users. This relates to the privacy notion of unlinkability which is also discussed by Pfitzmann and Köhntop; it refers to the inability to find a relation between two items in a system. In regards to key reuse, a key is obviously related to itself, but there will also be the relation between the transactions the keys are used in, allowing us to also link the transactions based on this. According to Pfitzmann and Köhntop unlinkability is one of three main concepts of anonymity and privacy. The other is anonymity, which is defined as such: "Anonymity is the state of being not identifiable within a set of subjects, the anonymity set." [14], and unobservability which means that no message is any different from random noise-i.e. encryption. The latter is not applicable to the Bitcoin system as it is a open system with a public record of messages (transactions) which is observable to the same degree by every participant. The system is also not anonymous as the subjects is distinguished by their keys.

Pseudonymity and unlinkability are the two privacy concepts used to avoid full transparency of activity within the system. A paper by Androulaki [15] suggests two privacy notions adapted to Bitcoin: activity unlinkability and user profile indistinguishability. The former refers to the inability of a observer to link keys or transactions belonging to the a single user of their choosing, and therefore revealing all the users activity. This is a more specified unlikability notion describing the lack of relationship between keys and transactions for a single user. User profile indistinguishability on the other hand is privacy for all users in the system. A user profile can consist of keys or transactions which are linked, meaning they are discovered as belonging to the same user. User profile indistinguishability means that a observer should not be able to link keys or transactions to the right user, and thus to construct correct user profiles. The linking of keys allows us to say that a set of keys belong to the same user, while the linking of transactions means that a user have participated in that set of transactions in some way. As noted by Androulaki et al. [15] key linking entails transaction linking because keys is contained within transactions; still there is room for transaction linking because other information might make us able to link transactions without being able to link keys.

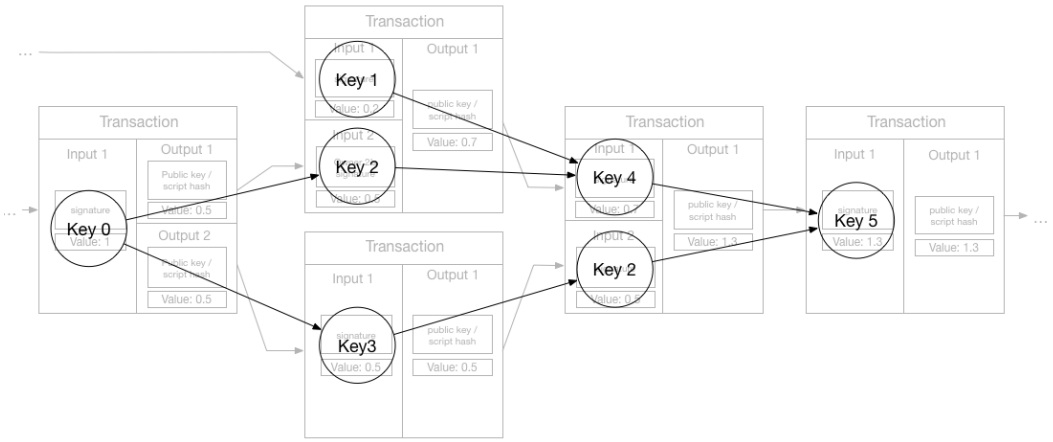

Figure 10: Transaction network with keys shown.

The research already conducted has focused on linking information found in the blockchain to negate the pseudonymous properties of Bitcoin, and thus create user profiles [5, 15, 4, 16]. To create profiles we need to link information that is related; this means linking related transactions to transactions, and related keys to keys. A simple way of linking transactions is using the fact that they are naturally related trough the output - input structure, creating a DAG as shown in Figure 1 and in the background of Figure 10. This graph will show us the relation between the transactions (nodes) connected with output - input (directed edges). However, this network will only show us how transactions are related and not much about the users, because the pseudonymous properties of the keys. This problem is illustrated in Figure 10 where we have overlaid the keys and the flow between them. We have seven total keys in this transaction sub graph, and with one key being used twice, the unique number of keys is six. Without linking keys the number of possible users will be the same as uniqe keys, as we can not easily determine if two keys is a uniqe user or the same user with a new key. Linking keys will create a set of keys controlled by the same user or entity, which means we have reduced the set of possible participants in the transaction graph. If we could link all keys related to a user, we would be able to find all activity that user participated in on the blockchain. Using the set of keys related to each user, we can create a user network similar to the transaction network, but where the nodes is users in addition to transactions, and the edges is the output-input pair from the transactions-i.e., showing the flow of Bitcoin from one user to another as illustrated by Figure 11. This user network shown in Figure 11 allows us to discover how founds is moving between users instead of just keys as was illustrated in Figure 10. Reid and Harrigan [4] created such networks in their paper analyzing Bitcoin privacy. In a paper reviewing Bitcoin privacy papers, Herrera-Joancomarti [6] states that this user network allowed Reid and Harrigan do user centered analysis, giving context to the data in the blockchain. Reid and Harrigan also created a ancillary network when linking keys (nodes being a uniqe key and edges being a relation between

two keys); This network would contain the key sets found for each user, each set represented by a maximally connected component in the network. A different type of user network was created by Meiklejohn et al. [5]: not structured as the transaction graph as the one in Figure 11, but instead showing users outside the context of the transaction graph. Each node is a user (linked key set) and the edges was any transaction between the two, showing us if a pair of users had done any transactions at any point in time. These networks illustrates the main goal of linking information: user profiles. A set of linked keys representing a user will allow us to see the activity of that user, and so we will able to make a profile for that user.

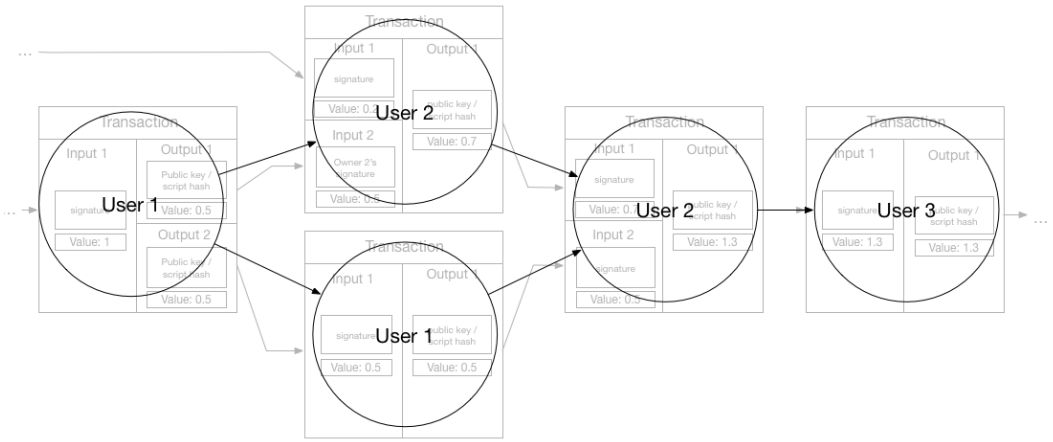

Figure 11: User network after linking addresses

The research done in this field has used two heuristics for linking keys. The first one has been used by several projects [4, 5, 15, 16] and was also pointed out by Satoshi Nakamoto in the Bitcoin whitepaper [2]. Keys used in transactions with multiple inputs can be said to belong to the same entity. As we explained in subsection Section 2.1.3: spending outputs requires signatures in the input, from the private key paired with the public key used to lock the output. A transaction with multiple inputs means that the all keys must have been available to sign the inputs, and so it is very likely that the creator of the transaction controls all keys used. If we consider Figure 10 we have two transactions with multiple inputs, and we can therefore link keys 1 & 2, and keys 2 & 4. As noted by Meiklejohn et al. [5] this effect is transitive, which means that by linking the keys from Figure 10 as done above, we can also link key 1 and 4, because both are related to key 2 which reused in the two transactions, so they must also be related to each other. By doing this we have a set of three unique keys controlled by one user; we can now use this set to define a user (user 2) in the user network as shown in Figure 11. The second heuristic utilizes "shadow addressees", first explored in [15] and then later refined in [5]. Because a output has to be spent in its entirety, a user wishing to transfer a value less than the value of available outputs must send the remainder back

to themselves. A real world example of this is receiving change when paying with a cash bill having higher value than the price of the item bought. Most Bitcoin software implementations known as wallets allowing users to do transactions, create outputs to transfer change back to sender automatically with a new key each time. Transactions doing this have multiple outputs: normal ones sending value to another entity, and a output giving back the change to the sender. The creation of this shadow address (key) and the locking of the output to this key is handled internally by the software, and users would not necessarily know the address, which makes it unlikely it is given to others for receiving transactions. This heuristic relies on the fact that these keys will only be used to receive the change in one transaction, and not in any other transactions. It also relies on people not using a single transaction to send value to multiple people, as the linking consists of finding multiple output transactions and assuming the shadow address is a key not used before in the transaction graph. Meiklejohn et al. [5] also notes that this relies on the implementation of wallet software, and is not a inherent property of the Bitcoin system itself. Applying this heuristic to the transaction graph with keys overlaid in fig.10 we will find one multi-output transaction with founds going from key 0 to key 2 and 3; key 3 has not been used before or after in the transaction graph while key 2 has, so it is very likely that the output to key 3 is change returned to the same user controlling key 0. Figure 11 shows the final result of the user graph, where we now have reduced the set of possible users from six to three by applying the two heuristics for linking keys.

The two heuristics discussed above for linking keys, uses information or characteristics from the Bitcoin system itself. However one can also link keys by using off-network information-i.e., sources outside the Bitcoin system. Reid and Herrigan [4] also used this approach and provides this example: exchanges allowing users to sell, buy, withdraw, and deposit Bitcoin will have access to the keys users utilize for depositing and withdrawing; which allows them to link those keys based on the user profile on that exchange. However, this information is not publicly available, but the method can be used for other types of services or data sources. People publicly disclosing multiple keys so they can receive Bitcoin is one such way. Reid and Herrigan [4] provides another example that is publicly available. A Bitcoin faucet is a webpage where people can donate Bitcoin which other users can receive a small amount of. The log of the redistribution containing ip-address of receivers is published to prevent abuse. This means that a reoccurring ip addresses can be used to link keys, similarly as an user profile. The information used for linking outlined above exists publicly on the internet, but one can also link keys by actively gathering this data. This was done in the study by Meiklejohn et al. [5], were they participated in mining pools, withdrew and deposited founds on exchanges or wallet services, used bitcoin gambling sites, and purchased goods from vendors. By doing this they were able to collect keys related to the different services and entities they did transactions with. This shows that engaging with other entities or users multiple times will allow us to link the keys used in those transactions. A important remark about this, done both by Meiklejohn et al. [5], and Reid and Herrigan [4], is that a big entity in the network which does transactions with a large number of other separable users will have greater opportunity to identify or track users, because they are able to use their central position to link keys. Such a entity could be a big ex-

change where people trade other currencies for Bitcoin, where by using the identifying information separating their users, and also the two heuristics discussed in the previous paragraph, they could link keys, create user profiles and thus track activity of many Bitcoin users.

Herrera-Joancomarti et al. discusses in their paper [17], privacy in relation to Bitcoin scaling solutions, and points to several problems with blockhain analysis. One central issue is the increasing size of the blockchain as it constantly grows. Herrera-Joancomartí et al. states that at the time of writing that paper the blockchain was 72 GB. The papers discussed earlier in this section, performed their analysis in 2013 when the blockchain was around 8 GB. At the time of writing this thesis, May 2018, the size of the blockchain is over 165 GB [18]. Herrera-Joancomarti et al. also points out that the linking heuristics used in previous studies are no longer applicable for off-chain transactions facilitated by networks like the LN. This is simply because the transactions will not be recorded on the blockchain. Herrera-Joancomartí et al. suggests that using payment channel networks will increase the privacy of users with regards to blockchain analysis, but that the existing methods may be adapted to analyze the payment channels in a network such as the LN. It has been suggested that identifying founding transactions on the blockchain can imply the network topology [19] of the LN. This is because all channels in the LN must be anchored in the blockchain with a founding transaction signed by both parties in the channel. If one where able to identify founding transactions on the blockchain, and also had some way of linking channels (represented on the blockchain as founding transactions) to each other, it wold reveal how the LN is structured as the linked channels would form the same network as the LN. This possibility is also mentioned in the LN paper itself [1] were they state that by monitoring the blockchain for founding transactions, a routing map of the LN can be theoretically built. Similarly, by linking the founding transaction and closing transaction of a channel, we get closed channels, so linking it with other channels would give us the structure of the LN in a previous state.

3 Methodology and Implementation

To meet our goal for this project as outlined in Chapter 1 we must create a method for identifying on-chain LN transactions within the blockchain, allowing us to discover what information is available in them. In Section 2.1 we covered how the LN uses the blockchain to operate with certain on-chain transactions. This means the on-chain LN transactions we will find are used to open, close, or claim founds related to a channel. As we are interested in what we can learn about the LN and its users from analyzing the blockchain, we will not utilize the information in non LN related transactions in this project. Reason for this is the size of the blockchain as discussed in Section 2.2, which means the requirements for both software and hardware will be high if all information in the blockchian should be kept track of when parsing it. Potential benefits of using information from all transactions, LN related or not, will be discussed in Section 5.2 on future work.

With the LN being layered on top of Bitcoin, it requires the Bitcoin system to work, while the opposite is not the case. The LN will also be aware of the state of the Bitcoin system, as it uses the Blockchian to operate the channels. The Bitcoin system on the other hand is unaware of the LN and does not distinguish between on-chain LN transaction and other transactions. This means that when we use the Bitcoin blockchian as the source of LN data, the amount of data will be smaller and not found in any LN context, compared to using the LN directly and collecting data from there. However, the LN is a dynamic changing network of nodes, channels, and payments; which is not stored in any public ledger like the blockchain. The information found in the LN can be recorded and stored by participants, but the system itself does not keep such a record. The data in the blockchain can be verified by any user as explained in Section 2.1.2, while data recorded from the LN could not be verified in the same manner on its own. This means that if we want verifiable historical data about the LN or to verify data collected from the LN, we must look to the blockchain. It will have limited and less information than what can be found in the LN, but will have the properties of being verified and automatically recorded by the Bitcoin system. While the Blockchain is the approach used in the project, there will be benefits from using the fact that there is more information available by connecting to the LN and record data. Collecting data from the LN will allow us compare it with what we find on the blockchain. By doing this comparison we can verify and quantify the effectiveness of our method for identifying LN relevant transactions on the blockchain. Comparing the blockchain and LN data can also show what information we are unable to get trough the blockchain, or to what extent something is available in the blockchain.

3.1 Blockchain analysis

The transactions in the blockchain is linked with outputs - inputs forming a DAG as we explained in Section 2.1.1. Parsing the blockchain involves linking these transactions to form this transaction graph, allowing each transaction and its data to be seen in context with every other. When applying the heuristics used by previous works discussed in Section 2.1, one can use the results of this to provide additional context to the graph-i.e., when users are defined by key sets, the pseudonymous properties of using new keys for each transaction is removed, and so user activity is revealed. Because we are interested in transactions related to the LN, we do not create a complete transaction graph containing all transactions on the blockchain; instead we only link transactions related to a single LN channel, which creates many small transactions graphs, each representing LN a channel, and each a sub graph of the complete transaction graph. We will refer to these graphs as channel graphs, because they contain the most common on-chain transactions related to a LN channel. By only creating channel graphs and not the complete transaction graph, we do not need keep track as much data during parsing of the blockchain, which makes our task of analyzing the blockchain easier. In our project we use two methods for locating possible channel graphs: using timelocked redeem script, and using 2of2 multisig redeem scripts. They are different, as the multisig method will get all potential channels, but will also give false positives, while the timelocked method is more precise in terms of identification, but will not be able to identify as many channels. Our focus has been on the timelocked method as because of the precision, but multisig detection has also played a role.

3.1.1 Using 2of2 Multisig for identification

Channel graphs created using this method will contain the founding and closing transactions as shown in Figure 12. This will also be the case for all types of channel graphs, as it is these transactions that represents a channel on-chain. To create the channel graphs when parsing the blockchain, we must be able to differentiate between on-chain LN transactions and other Bitcoin transactions. Therefore, we must find some characteristics needed or only found in on-chain LN transactions. In Figure 14 we can see some characteristics of the founding and closing transactions. The output in the founding transaction used for the channel must be `P2WSH`, and the closing transaction will always have one input which will be a 2of2 multisig redeem script as explained in Section 2.1.5. The different characteristics have different levels of uniqueness, but using more unique characteristics will often impact the number channels which can be identified, meaning there is a trade-off between precision and potential results. An example of this is the `P2WSH` output type, which is not very unique, so it will provide many false positives, but will allow for detection of both closed and open channels, which we will discuss more in Section 3.1.4. The 2of2 redeem script is on the other hand more unique, as it is a specific redeem script and must therefore also belong to a `P2WSH` output - input pair. In the founding transaction we will only be able to check if it has any `P2WSH` outputs, while the closing transaction will allow us to check for the 2of2 multisig characteristic, which the presence of also entails the `P2WSH` characteristic. Using closing transactions will limit us to closed channels, but the transactions matching our characteristics are more likely to be LN related. While

the 2of2 multisig reedem script will be present in all closing transactions for LN channels, they can also be used for other purposes. We therefore cannot use this to determine with certainty if a transaction is a closing transaction or not, but it will rule out all transactions without this characteristic. So by locating all transactions having inputs with such redeem scripts, we will have all potential closing transactions, and by following the input - output connection we can also locate the founding transaction, providing us with a channel graph with the transactions as shown in Figure 12.

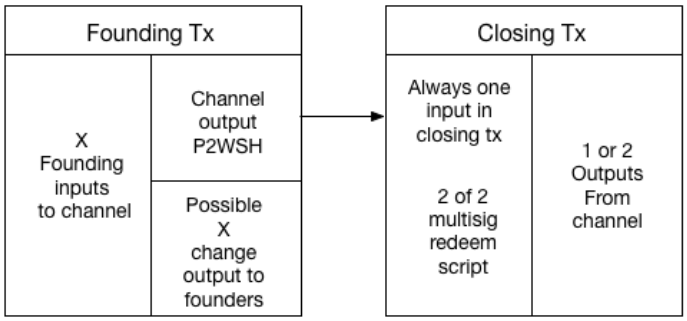

Figure 12: Channel graph with founding and closing transactions

Creating the channel graphs as explained above, identifies the transactions in the channel graph in reverse in relation to their creation order-i.e., we identify the closing transaction first, and use it to find the founding transaction. As stated before the transactions form a DAG, meaning they will be located chronologically in the blockchain, with newer transaction being found towards the tip of the chain. For the transactions in our channel graphs this means that the closing transactions will be found before the founding transaction, as the closing uses the output of the founding. We must therefore parse the blockchain in reverse, beginning with the most recent block, and working towards the genesis (first) block. This will ensure that we encounter the transactions in the order we want-i.e., closing transaction before the founding. As explained in Section 2.1.2 the blocks in the blockchain contains the hash of the previous block, which creates the chain connecting them. This provides a way of easily parsing the blocks and therefore also the transactions in the desired order.

The software we have developed for this project parses the blockchain and identifies relevant transactions, using both the multisig and timelocked identification methods. We use the btcd [20] Bitcoin implementation written in Go, to synchronize and store the blockchain data. Our software, which is also written in Go, uses libraries from btcd to read this data from disk and make it available to us in a convenient manner. At a high level our software finds the latest block from the blockchain stored on disk and uses it as a starting point; it then iterates over the transactions in the block, and checks if they are relevant for our project-i.e., is part of a channel graph. If that is the case they are stored in a data structure representing a channel graph. After all transactions in a block is parsed,

the hash of the preceding block is used to fetch it from the disk, and the same process is repeated. For each transaction parsed, the input part is checked for the presence of a 2of2 redeem script, if that is the case, a channel graph is created and the transaction is stored as a closing transaction. As explained in Section 2.1.4 each transaction has a id, which is the transaction hash. Transaction inputs references the hash of the transaction containing the output they spend, so when finding a potential closing transaction we get the hash of the corresponding founding transaction. This is stored in a list, which we can use to recognize founding transactions when we encounter them in the blockchain. This means we always check the hash of each transaction we parse, to see if it is the founding transaction for a closing transaction we have already found. If there is a match, we have found all transactions for a single channel graph. This is the main algorithm for the software as can be seen in Figure 13, where the process described above is illustrated.

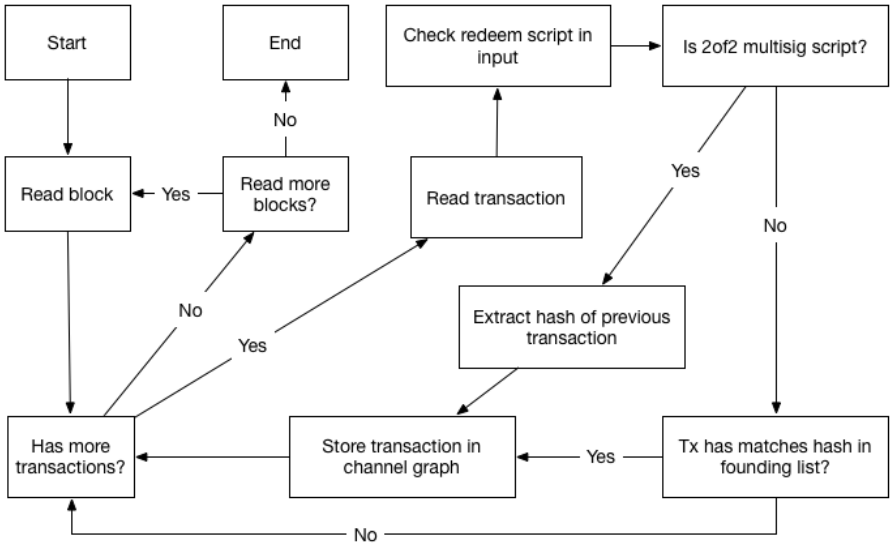

Figure 13: Algorithm for parsing blockchain and creating channel graphs using the multisig method

The 2of2 multisig redeem script used to identify possible closing channels is shown below. Also a example of the byte version of such a script included, as this is how we get the script when extracting it from the input of a transaction in our software, and therefore use when identifying the script.

```
2 <public key 1> <public key 2> 2 OP_CHECKMULTISIG
```

[82 33 2 211 153 245 240 225 125 95 140 116 20 99 81 38 139 135 136 59 14 125 34 181 148 47 67 16 42 24 147 28 144 61 33 33 2 215 1 70 141 233 112 91 253 252 202 27 73 158 254 234 159 125 98 30 78 159 235 6 46 167 103 105 239 180 125 168 66 82 174]

As we explained in Section 2.1.3 the operations in Bitcoin scripts is called opcodes, and the Bitcoin wiki contains a list of all of them, and their encodings [21]. The opcodes and their ordering in the script dictates the functionality of the script, so to identify a type of script we should look for scripts having the same format and operations. We covered in Section 2.1.3 and Section 2.1.5 the purpose of this script, which to create a scenario where multiple keys is required to unlock a output and spend it. The scripts consists of of two numbers denoting the required and total number of keys required to create signatures, the public keys for the key pairs that can be used, and the `OP_CHECKMULTISIG` operation. The required and total number of keys required will in our case be 2, as we are only looking for 2of2 scripts. As shown above, the script starts with the required number of keys which must be used for signing, in this case 2, being 82 in the byte version. Then there is the first public key of the key pair able to sign. In the byte version the second vector is 33, which indicates how many bytes to push to the stack; this is the length of a compressed public key with prefix. Bitcoin uses elliptic curve cryptography, where a public key is two coordinates representing a point on the curve. The prefix for compressed keys can be 02 or 03, indicating if the y coordinate is even or odd [8]. So the first byte vector in a compressed public key, and the third byte vector in this type of script will always be 2 or 3, and the 32 next bytes will be the rest of the compressed key `<public key 1>`. Then we have the other public key `<public key 2>` in the same format, followed by 2 or 82 in byte format, telling us the total number of keys which can be used for signing. At the end we have the opcode 174 `OP_CHECKMULTISIG`, which checks signatures against the public keys. To identify this type of script we check if the opcodes are present at their expected locations in the script, and the total length of the script. This means we check if certain indexes in the byte array contains the data we expect-e.g., we would expect the last index in the byte array to contain 174.

When using 2of2 multisig redeem scripts, which also entails using `P2WSH` for identifying potential channels, our result will contain all actual LN channels, but likely also many that are not. The effectiveness of this will depend on the current use-cases for 2of2 multisig transactions besides on-chain LN transactions, and how common these are. To determine the current effectiveness of this method, we will compare data from the blockchain with data gathered through the LN, and with results from our other method. This will allow us to quantify the actual number of channels identified using this method, compared to the amount of false positives. The Section 3.2 will contain a more in-depth discussion of this.

3.1.2 Identification using timelocked redeem scripts

The main method for identifying channels we have used in this project, relies on timelocked redeem scripts. While being more unique than the 2of2 multisig redeem script, it will only exists on channels that is unilaterally closed-i.e., the channel was closed by one of the parties, and not cooperatively. In Figure 14 we can see how a channel graph is structured when this method is used. We still have the founding and closing transactions, but the closing transaction is also a commitment transaction, as it was used to close the channel. When a commitment transaction is published the output to the entity closing the channel will be timelocked, as we explained in Section 2.1.5. This is to enable

the other party to spend the output using the revocation key, in the case of the commitment being revoked-i.e., any commitment other the most recent. This is implemented using a `P2WSH` output, which have a redeem script containing a timelock, and a clause for the revocation key. As redeem scripts is only available in the input of the transaction spending the `P2WSH` output, similarly as with the 2of2 multsig redeem script, we will find this script in the transaction spending the timelocked output. We will refer to this transaction as the timelocked transaction, which can be seen on the right in Figure 14.

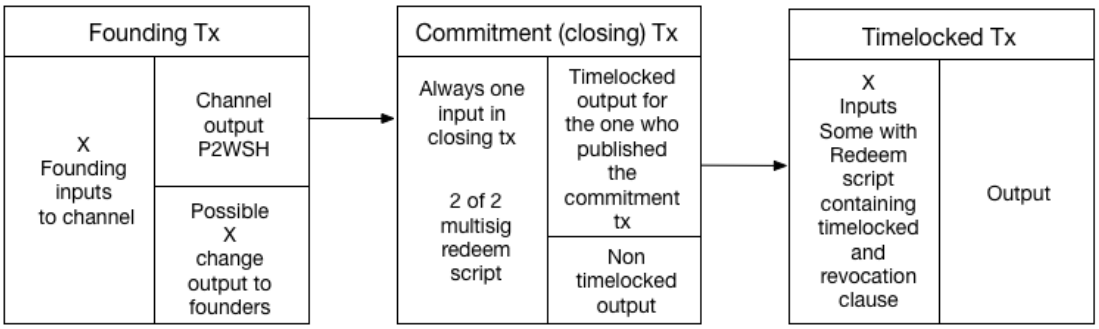

Figure 14: Channel graph of unilaterally closed channel

While the timelocked transaction containing the redeem script will not be present in channels closed cooperatively, the uniqueness of the the script will ensure the transactions that is identified using this characteristic are very likely to be actual timelocked transactions from the LN. When a timelocked transaction is identified, we can use its input where we found the redeem script to get the hash of the closing/commitment transaction. We can again use the input of the closing/commitment transaction to get the founding transaction. This way we be able to get all transactions in the channel graph. We can also use the fact that the closing/commitment transaction will have the 2of2 multisig redeem script. By checking that the closing/commitment transactions has this script in its input, same as we did in the multisig method, we are further ensuring we have found a LN channel. Even with the timelocked scripts being very unique, because of its specific use case, users are free to create the scripts they wish, so the presence of such a script will not guarantee that we have found a LN timelocked transaction. While also LN related, we will see in Section 3.1.3 how scripts having the same purpose being used in different transactions. But by also checking that the closing transaction has the characteristics we expect, we can avoid mistaking instances where a timelocked script used in other scenarios for timelocked transactions.

The identification and subsequent creation of channel graphs using this method, is implemented in our software in the much the same way as we described for the multisig method in Section 3.1.1. To reiterate, timelocked transactions is identified using redeem scripts in their inputs, then the

transaction hashes of the inputs is used to find the closing/commitment transaction. The input of the closing transction is checked for the precence of 2of2 multisig redeem scripts, and if present the transaction hash of the founding transaction is used to locate it. Each channel graph is created incrementally as we find each transaction it contains. Instead of only having a list containing hashes of founding transactions, this method will also have a list of hashes for closing transactions. For each transaction we parse, we first check if it is a timelocked transaction, if not, we check if its hash matches any found in the hash lists-i.e., check if it is a closing/commitment or founding transaction. The lists are created as we locate timelocked and closing/commitment transactions and find hashes of previous transactions. This whole process can be seen in Figure 15, which is similar to the one in Figure 13 but more complex.

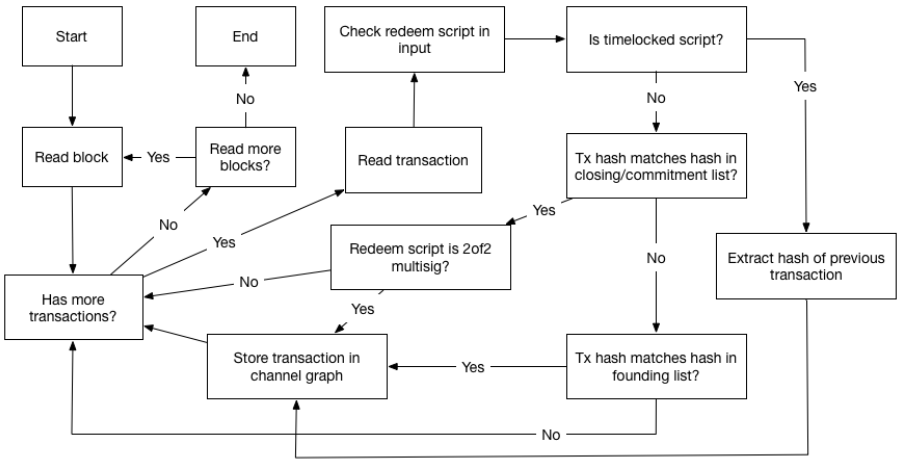

Figure 15: Algorithm for parsing blockchain and creating channel graphs using the timelocked method

The timelocked redeem script defined in [13], is the redeem script used to identify the timelocked transactions, and can be seen below:

```
OP_IF
    # Penalty transaction
    <revocationpubkey>
OP_ELSE
    ‘to_self_delay‘
    OP_CSV
    OP_DROP
    <local_delayedpubkey>
OP_ENDIF
OP_CHECKSIG
```

The script contains a if - else clause, with the if clause being the one which allows the opposing

party to spend the output in the case of the commitment transaction being revoked. In Section 2.1.5 we explained in detail how the parties will revoke old commitment transactions by exchanging keys, allowing the other party to spend the timelocked output with the key, and thus claim all the founds in the channel; this first clause is how this is done in practice. By supplying `<revocation_sig> 1` with a valid signature, the script will evaluate to true. The 1 will make the script evaluate the first clause, and then the signatrue will be matched to the `<revocationpubkey>` using `OP_CHECKSIG`. The else clause is the timelocked portion of the script. It will contain the timelock, enforced by the `CHECKSEQUENCEVERIFY` operation defined in [22], which will terminate the script if the specified delay has not passed. The delay is then removed from the stack with `OP_DROP` such that a signature and public key are the only two items on the stack. To make the script use the else clause with the timelock, and check the signature against the `<local_delayedpubkey>` the witness data should be: `<local_delayedsig> 0`. Similar as to the revocation clause we have a signature, and the 0 will make the script not use the if clause. Meaning the `OP_CHECKSIG` will used to verify the `<local_delayedsig>` against the `<local_delayedpubkey>`, as long as the timelock has expired. This means we can easily check if a timelocked script was revoked. By checking the witness data provided to the script for 0 or 1, we can determine which clause was used, and as it is included in the blockchain the script has been evaluated and found to be valid. Below is the raw byte representation of a specific redeem script of the type shown above. Again, this is because our implementation handles the byte version of the scripts when identifying them. The byte version has been formatted in the same manner as the clear-text version, to allow for easier comparison.

```
[99
   33 3 251 83 243 198 231 109 204 252 217 94 44 221 0 255 185 86 106 105 161 141 254 96
   167 77 48 16 57 146 128 4 80 1
103
   82
   178
   117
   33 2 159 6 236 212 233 63 48 147 59 52 201 11 15 138 165 248 118 100 188 234 227 215
   108 160 135 22 57 37 117 250 172 130
104
172]
```

The first byte vector seen in the byte script above is 99 which is the opcode for the the `OP_IF` operation. The next vector is 33, which indicates how many bytes to push; As explained for the 2of2 multisig script: this is the length of a compressed public key with prefix. So the 33 next vectors will be the `<revocationpubkey>`. After this public key we will find the `OP_ELSE` with the opcode 103, followed by the 82 'to_self_delay' which is the delay used for the `OP_CHECKSEQUENCEVERIFY` (`OP_CSV`) operation, having opcode 178. Next vector is 117 which is the `OP_DROP` operation, and after that we find the `<local_delayedpubkey>` in the same format as `<revocationpubkey>`; 33

indicating the number of bytes to push to the stack followed by the prefix and the rest of the key. After that we have 104 which is the opcode for `OP_ENDIF`, and 172 for `OP_CHECKSIG`. To identify this type of redeem script in timelocked transactions we check if the opcodes can be found at the expected locations. E.g., index 0 of the byte array should be 99 for the `OP_IF`, and 103 for the `OP_ELSE` operation should be at index 35, allowing space for a public key between it and the `OP_IF` operation. Each opcode found in this script type is used for recognition, meaning all opcodes covered here must be present in the script for it to be identified as a timelocked redeem script, and they must also be found at their expected location relative to the others.

As stated before, the inclusion of transactions in blocks naturally orders them, because transactions added in new blocks must have inputs from transactions already in the blockchain. However, transactions pairs where one spends another can be confirmed at the same time, meaning they are included in the same block. This occurs because the creator of the latest transaction spends a unconfirmed transaction-i.e., a transaction that has not yet been accepted as part of the shared transaction history of the network. Transactions are not necessarily ordered within blocks, so if two sequential transactions are located in the same block, we might encounter the latter transaction first. This means we might encounter the founding transaction before the closing, but since we have not yet found the closing transaction, and therefore did not know the hash of the founding, we could not yet recognize it. Because if this we had to iterate over the transactions in the block a second time if we found any transactions, to make sure we did not miss the preceding transaction earlier in the block. Even with parsing some blocks twice the software parsed blocks fairly quick, parsing 100 000 000 transactions in under 90 minutes. The memory consumption were also low, as our software only stored channel graphs and the hash lists for transactions we where currently looking for.

3.1.3 HTLC on-chain

In Section 2.1.7 we covered how transfers across multiple channels is facilitated by using the HTLC construct. As explianed, if the HTLC cannot be resolved off-chain, the channel will be closed and the HTLC will be handled on-chain. This will be a `P2WSH` output from the commitment/closing transaction, with a redeem script allowing users to spend it in two ways: by either supplying the preimage R used to create the HTLC, or by waiting for the timelock to expire, which will create a additional timelock. There are two versions of HTLC redeem scripts used in on-chain transactions, defined in the BOLT rfc [13]. Which of the two types is used depends on if the party that published the commitment transaction to the blockchain is sending or receiving founds using the HTLC. These scripts are even more complex than the timelocked scripts, which would make them very good at identifying LN related transactions. Below is a the script used for the HTLC output when the publisher of the commitment has sent a HTLC to the other party:

```
# To remote node with revocation key
OP_DUP OP_HASH160 <RIPEMD160(SHA256(revocationpubkey))> OP_EQUAL
OP_IF
```

```
    OP_CHECKSIG
OP_ELSE
    <remote_htlcpubkey> OP_SWAP OP_SIZE 32 OP_EQUAL
    OP_NOTIF
        # To local node via HTLC-timeout transaction (timelocked).
        OP_DROP 2 OP_SWAP <local_htlcpubkey> 2 OP_CHECKMULTISIG
    OP_ELSE
        # To remote node with preimage.
        OP_HASH160 <RIPEMD160(payment_hash)> OP_EQUALVERIFY
        OP_CHECKSIG
    OP_ENDIF
OP_ENDIF
```

Without going into great detail, the script first checks if it was provided the public revocation key. If this is the case, the first if clause will be used to check if the signature provided matches this public key. This will only happen if the HTLC was part of a revoked commitment transaction. If this is not the case the else clause is used, which will check if the HTLC public key of the party not publishing the commitment was provided. If this key is provided it means that party can use the preimage `R`. The publisher can make the HTLC timeout by using the notif clause. The other script is structured different, but has basically the same functionality as it swaps which of the parties can use the preimage `R` or timeout the HTLC.

Our software is able to recognize these scripts using similar methods as discussed in Section 3.1.1 and Section 3.1.2. They are however not used for identification of channels, and thus channel graph creation. Our focus has been on timelocked transactions as the main method for identification, as it provides a good mix of precision and amount of results. Our theory was that HTLC outputs, while being very unique, are not nearly as frequently found on the blockchain as timelocked transactions. Additionally, the transactions containing redeem script like the one above, would spend commitment/closing transactions we would likely find using timelocked transactions. But by implementing the basic recognition of the scripts, we can check how often they do occur, and by comparing their inputs to our already identified closing transactions we can determine how many of the HTLC transactions would identify channels we could find using the timelocked method.

As mentioned when explaining the script, and described in Section 2.1.7, the transactions spending a HTLC output will have a additional transaction with a timelock if the party that published the commitment spends the HTLC. This timelock will use the same script as the timelocked transaction spending a commitment output. In Section 3.1.2 we mentioned how after identifying a timelocked transaction, we would check if the preceding transaction had a 2of2 multisig script, indicating it was a closing transaction. Not doing this, we would create channel graphs where the timelocked transaction was a HTLC timelocked transaction, the closing transaction a HTLC transaction, and the

founding transaction a closing transactions. This example shows that a specific script does not necessarily mean the transaction is the one we expect. In Section 5.2 we will discuss some possibilities by using HTLC related transaction and the data they contain.

3.1.4 LN information on the blockchain

If we can successfully identify channel graphs containing transactions related to a LN channel, the data in those transactions will allow us to extract some information about the channel. The value in the output - input pair used for the channel, will tell us the total value of the channel. Timestamps is included in blocks when they are created, so by checking the timestamp of the blocks where the founding and closing transactions is located we can see how long the channel was active. The number of inputs in the founding transaction and the number of outputs in the closing is also interesting; a founding transaction with a single input shows that a single user founded the channel; multiple inputs can indicate that both users founded the channel, but there is also a possibility that the channel is still founded by a single user, having multiple smaller outputs to get the desired value of the channel. It is difficult to determine how founds have moved inside the channel based on the value of the inputs to the founding transaction, and the value of the outputs from the closing transaction-i.e., it is hard to extract any information about any off-chain transactions by analyzing the outputs and inputs of the on-chain transactions. The reason for this is that normally we can not determine which input/output belong to which user in the channel. We discussed in Section 2.1.3 how keys are used to lock outputs, and signatures using those keys are used in inputs to unlock, meaning a key pair is related to a input-output. In Figure 16 we have a founding transaction with two inputs (keys), a multisig output - input with two keys, and a closing transaction with two outputs (keys). If we assume each user founded the channel with one input we can determine the initial balance between the parties by checking the value of those inputs. Comparing that balance to the one in the two outputs will tell us how it has shifted from start of the channel to the end, but as discussed in Section 2.2 the pseudonymity provided by different key pairs makes us unable to see which of the inputs corresponds to which output. We can also see in Figure 16 how the value is initially spread in two outputs then merged in the channel and then spread out again in two outputs, so simply following a specific value will not work. Linking the keys used in these transactions as is done in previous work discussed in Section 2.2 will make this possible and is something we will discuss in Section 3.3. The one thing we can determine by looking at inputs and outputs of a channel, without any key linking, is that a off-chain transaction has taken place in the channel, but this is only the case if there is one input to the funding transaction (single founded), and there is multiple outputs, meaning we know the channel started with with all value belonging to one party and it ends with the parties splitting the value.

We discussed in Section 3.1.1 how using `P2WSH` outputs would allow us to identify possible open channels, as all outputs used for LN channels are of this type. With the characteristic being fairly common we will most likely have many false positives, but we can nevertheless use this to determine the maximum size (number of channels) of the LN. By counting the number of unspent

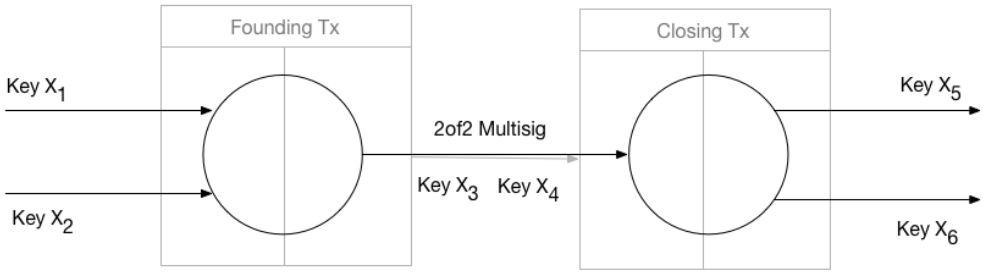

Figure 16: Keys in a channel graph

P2WSH outputs we will get maxium number of channels open. This can also be done to get the maximum number of LN channels at any historical point of the blockchain; by counting all P2WSH outputs and subtracting the spent ones up to a height on the blockchain will get us the unspent outputs at that point. While this might not be accurate in terms of actual number of channels (size) for the LN, it will give us a concrete upper limit to its size, both current and at any point in the past. We can also use data collected direclty from the LN to correlate between the number of unspent P2WSH outputs and open channels in the LN at different points in time. Which will indicate how closely the number P2WSH outputs reflects the number of LN channels.

3.2 Lightning Network data collection

This project focuses on exploring the LN and related data trough the blockchain, but we can also use other approaches to verify and provide context to our view from the blockchain. One such approach is simply to collect data from the LN directly. This data will be more complete than what we can find on the blockchain, so it will provide us with the bigger picture. By comparing blockchain data with data directly from the LN, the effectiveness of our methods can be measured. In Section 3.1.1 discussing our multisig detection method, we stated that it would create some channel graphs which where not actual LN channels. By comparing the potential channels we find using this method with the ones gathered directly from the LN, we can determine the how many of our results are false positives. The data from the LN can also provide us with the ideal result of linking information found on the blockchain. This is because some relations are explicitly defined in the LN data, but only implicitly or not at all on the blockchain, which makes our linking efforts on the blockchain data also measurable.

To collect data from the LN we used the one of the implementations following the BOLT specification [23], called LND [24], which is a open source project written in Go. We modified the implementation to collect the data we where interested in. LND maintains a view of the current LN by storing a graph containing active channels and nodes. This is continuously updated as new channels are announced trough the network, and closing transactions is found on new blocks on

the blockchain. It requires a Bitcoin instance running to interact with the Bitcoin network, which is used to publish on-chain transactions and monitor the blockchain. New channels are discovered by announcements within the LN, while channels closing is found by locating closing transactions in new blocks from the Bitcoin network-i.e., transactions spending outputs of founding transactions. We modified the 0.4.1-beta version of LND to create a copy of its database each time a new block notification was received from the Bitcoin software. This was not done immediately but within a minute so the changes from the block could be applied to the graph data. Doing this gave us a set of databases containing the state of the network at each block, essentially a set of snapshots of the LN.

The snapshots contains the graph describing the state of the lightning network as known to our node at the time of the snapshot, so by comparing the graphs of different snapshots we can see how the network evolves over time. Each graph in the snapshots is essentially a set of channels active at that time. Consider two sets of channels representing the LN at different points in time: κ being the older and τ the newer. The channels not present in κ but present in τ would be new channels-i.e., the relative compliment of κ in τ, $\tau \setminus \kappa$. Similarly the channels closed would be the ones present in κ but not in τ-i.e., the relative compliemnt of τ in κ, $\kappa \setminus \tau$. Doing this for each snapshot we can get a set of channels closed and opened during the data collection interval, and because we collect snapshot at each new block, we can easily contextualize the results with different block heights-e.g., create a list of closed channels for each block, or number of active channels for each block height. This also makes it easy to compare data from the LN to data from the blockchian as we can use the block height as a index, ensuring we compare correct data. Using some libraries from the LND implementation, we created a additional piece of software to read the data generated by the node software, compare the snapshots, process the data, and produce outputs with the desired information.

One modification we made to the LND implementation besides generating snapshots for the network state, was removing the process of pruning the "zombie" channels from the graph. According to the BOLT rfc [25], nodes can remove channels if the last channel update is older than two weeks. This is done to avoid open but abandoned or unusable channels to remain in the graph and be propagated to others. The LND implementation follows this recommendation and does this regularly. Our reasoning behind disabling this for our node was to keep data from the LN consistent with the blockchain data. Pruning channels would make changes to the LN graph, making it seem like channels where closed when they from the blockchain perspective where still open. Because we are focusing on the blockchain in our project, and is not interested in the routing availability of channels in the LN, we choose to keep these channels in the graph.

3.3 Linking and key reuse

Linking or clustering information that is related has been the focus of much of the related work described in Section 2.2. The main goal when doing this is to reveal non-explicit information, which in the context of the Bitcoin transaction graph would be constructing user profiles. This is done by

linking keys controlled by the same user, which will counteract the pseudo-anonymous properties of the Bitcoin system. In our project we are only interested in the subset of Bitcoin users that is also LN users, in addition to information about the LN itself. Our methods for locating this relevant information have been outlined in Section 3.1, the result of which is channel graphs containing the on-chain transactions related to one LN channel. As we do not take the entire Bitcoin transaction graph into consideration we are limited to using the data found inside channels graphs for linking. The main focus of linking in this project is to find relations between channel graphs, allowing us to link them based on shared user participation. It relies on the fact that to be a usable network, the LN will need some nodes/users to have multiple channels to different users-i.e., some vertices will need to be of a degree higher than one. This relation between channels is clearly visible from the LN perspective, as the view of the network will contain this relation, but this is not the case for the blockchain. As we stated in the start of this chapter, the Bitcoin system is unaware of the existence and state of the LN, meaning the there is no explicit relation based on common nodes/users participation for channel graphs on the blockchain. As the on-chain LN transaction are mainly for managing channels, we are limited to linking information about channels or users within the channels. The off-chain transactions will by their very definition not be available to use for any linking.

While we can link channels based on a common user participating on both, we can not determine which of the two users in either channel creates this relation. The reason for this is the same as discussed in Section 3.1.4, where we pointed out the problems of matching inputs to outputs of a channel, for seeing changes in balance between the users. In previous works we discussed in Section 2.2 users where defined by control of a key pair, so essentially a key is a user. Linking keys will reduce the number of users by redefining users as a set of key pairs. E.g., a transaction graph with four keys, would mean there was four potential users, unless we managed to link keys which would reduce this. For our case with the channel graphs, a problem arises because we know in advance how many users is involved in the graph. There is two users involved in a channel, but usually many more keys present, so we have a reduced user count, but this is not due to linking keys. Defining a user in this setting is harder, as we cannot simply choose a arbitrary key for each of the two users. In Figure 16 we see how there are 6 keys which normally could each represent a user, but in our setting we know there is only two users. If we could link the keys, resulting in two sets of keys, then these sets could be our definition of a user as it would enable us to distinguish the users. But without two key sets, when linking channels based on a property in the channel graph, we cannot determine which of the users this property belongs to as we have no clear definition of the users. We will however discuss later in this section some possibilities for defining users.

We have created three heuristics for linking channel graphs in this project. These are only for linking channel graphs based on common user participation; two of them can however be related to the heuristics discussed in Section 2.2 used in previous work. The heuristics employed are:

1. **Key reuse**

This heuristic is simply to check if the same keys is present in different channel graphs. Any key found within a channel graph is sure to belong to one of the two users participating in the channel, so if the same key can be found in another graph we know the user owning the key participated in both channels. This heuristic becomes useful because we are linking sets of transactions-i.e., transaction graphs representing channels. In previous work where they linked keys, this becomes redundant as a key is obviously related to itself, but in our case we are not interested in keys as such but rather how they reveal user involvement. Because we know there is only two people participating in transactions within the channel graph, keys from any transaction inside the graph can be checked for reuse. The keys used in the founding and closing transactions clearly belongs to the participants, but with the timelocked transaction we cannot be so sure. The BOLT rfc [26] heavily recommends the owner to spend the timelocked output when it expires to transfer the value to a convenient address, this is because of the advanced redeem script required to spend the timelocked output. This advanced script must be kept until the output is spent, which is inconvenient compared to having the value available in a normal `P2WPKH` output. Assuming implementations follow this recommendation, the timelocked transaction and the keys found within should only contain keys related to the user which claimed the timelocked output. We should however note that the key reuse needs to happen in different channels, as key reuse within a single channel does not allow us to link any channels.

2. **Graph connections**
   This heuristic is based on channel graphs being directly connected with each other by output - input pairs. Such a connection can be found in any outputs going to transactions outside the graph. This means all outputs in the founding transaction except the 2of2 multisig used for the channel, the non timelocked output in the closing transaction, and the outputs of the timelocked transaction can all be used for linking. In Figure 17 we can see these connections-e.g., the non-channel output of a founding transaction is used for input in another founding transaction. The reason we can determine shared user participation in connected channel graphs is that they are directly connected. By inferring the purpose of outputs from the channel graph we can determine that the outputs are controlled by the participants in the channel, so if they connect to another graph we can say there is common user participation. This concept is the same as the one used in previous work with heuristic 2 discussed in Section 2.2, where some outputs from a transaction was assumed to be change for the sender. The non channel outputs of the founding transaction should be change back to the participants, as they may not want to use the entire value of the inputs in their channel. It is unlikely that the outputs is a transfer of funds to another entity unrelated to the channel, so we can safely assume that the outputs are controlled by one of the entities in the channel, meaning if they are used as input in another channel, it would tell us that the same user is participating in both. For the non timelocked output in the closing transaction, we know it must be controlled by one of the users as it is how the channel works. The control of the outputs of the timelocked transaction

is based on the recommendation from the BOLT rfc [25] which was discussed in the previous heuristic.

3. **Graph overlap**
   The third heuristic is when channel graphs overlap with each other, meaning a single transaction is part of multiple graphs. This can only occur with the timelocked transactions, as founding and closing transactions are used for controlling a single channel. This can change however, as we will discuss further in Section 5.2 on future work. We can see this overlap illustrated in Figure 17 where the timelocked transaction in the top right has inputs from two commitment/closing transactions and will thus be part of both channel graphs. This relates to heuristic one used in previous works as outlined in Section 2.2, where transactions with multiple inputs was used to link keys used in those inputs. By being part of multiple graphs, a timelocked transaction will have multiple inputs. Which according to the old heuristic 2 tells us that the same user in control.

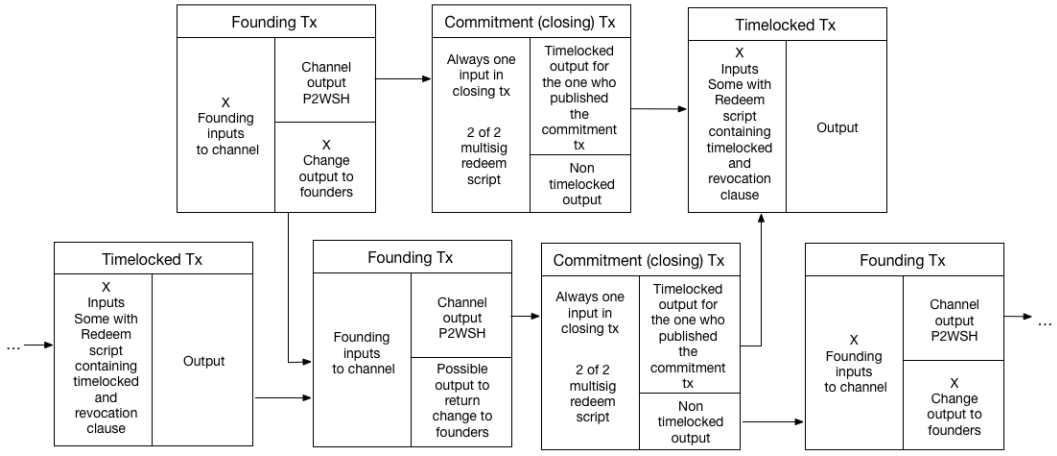

Figure 17: Connected channel graphs

Linking channels based on shared user participation allows us to create a channel network visualizing how the channels are related to each other. In this graph the vertices represents channels and the edges represent users/nodes in the LN. While the transaction graph for Bitcoin allows us to see how founds are moving by giving us the structure of the transaction graph, the channel network can show us how founds can flow between channels based on their connection by users. As the LN is a network itself, it has a natural graph structure where vertices are users/nodes and the channels is edges between the users, which we will refer to as the LN graph. This is the reverse in terms of vertex - edges compared to our channel network. Another difference between the LN graph and our channel network, is that the LN graph requires distinction between users/nodes as the vertices in that graph is a specific user; in our channel network the edges is any of the two users participating in the vertices connected to the edge.

Having a clear definition of users in the context of channels identified on the blockchain, will as discussed allow us to differentiate the users, meaning we can determine which channels a specific user participates in. This would allow us to crate a network using channels we have identified on-chain, having the same structure as the standard LN view-i.e., vertices representing users and edges being channels. Including the temporal aspects of the channels would allow one to recreate the structure of the LN at different points in time. One approach to achieving a clear distinction between the two users in the channel graphs is to disregard keys and related transactions; if we only use the 2of2 multisig output - input constituting the channel, we will only have two keys, one for each user. While this removal of information gives us a clear definition of a user in the channel, it also removes many possibilities for linking the channel to other channels. The only heuristic still viable if we only consider the channel output - input, is key reuse of the keys used to define the users. This means we will not be able to link as many channels as previously, where we would have more keys to check for reuse, in addition to two other heuristics. Another possibility is to link all keys within a channel graph, resulting in two sets which each can represent the user as we discussed previously. This approach would allow us to use the heuristics presented above and also differentiate user participation. A hybrid between the two would also be possible: linking keys to the extent we are able to, and then disregarding information related to the keys we are unable to link to any set. The heuristics could still work using this approach, depending on the extent we are able to link keys, and which keys we are able to link. Linking keys should be done by using the entire transaction graph, as doing this will provide more linking possibilities than only taking the few transactions inside the channel graphs into consideration. This will be discussed more in Section 5.2 on future work.

4 Results

In this chapter we will present our results from analyzing the data in the blockchain using the methods described in Chapter 3. Bitcoin has a testnet used for development and testing of experimental functionality, which the LN have been tested on for longer than it has been used on the mainnet. We have used both the testnet and the mainnet in the project; our reasoning for this is the LN on the testnet is larger, and people are testing a wide array of use cases and functionality, using testnet coins without any value; on the mainnet users have Bitcoin with value, perhaps resulting in more realistic user behaviour which could impact the data generated. By using both Bitcoin networks we will have both edge cases and cases accurately reflecting normal user behaviour. The testnet has its own blockchain which is publicly available in the same manner as the mainnet blockchian. Collecting data from the LN as discussed in Section 3.2 was done in two intervals of one week each. Creating a snapshot of the LN state on each block for a week results in around 1000 blocks worth of data. This should be sufficient to verify our methods of identifying relevant LN transactions, and doing this twice allowed us to make adjustments after the results of the first interval.

4.1 Method verification with LN data

Comparing data from the LN with our data from the blockchain, provided us with results about the extent our identification methods are able to identify LN channels on the blockchain. There was three sets of data used in these comparisons:

- The set α , containing channels from the LN closed during the capture interval.
- The set β, containing channels from the blockchain identified using timelocked redeem scripts,
- The set γ, containing potential channels identified on the blockchain using 2of2 multisig scripts.

In Section 3.1.1 we discussed how the multisig method could be used to give us a set of potential transactions being related to LN channels. Because the on-chain channel transactions must be of this type, the γ set is guaranteed to include all LN channels closed in the block interval used for the search. It also means that the two other sets will be subset of γ. So the relations between the sets should be as follows:

$$\alpha \subseteq \gamma, \quad \beta \subseteq \gamma, \quad \beta \subseteq \alpha \tag{4.1}$$

The two first relations is explained above and is always true for the systems in its current state. The last relation however, will not hold with our data, but ideally it should be true. For this to be the case our method with timelocked redeem scripts for identifying channels, should not have any false positives-i.e., identify channels which is unrelated to the LN. Additionally we should also be able to

discover all closed channels in our interaction with the LN during the interval, meaning we would get the complete set of channels-i.e., γ contains all LN channels closed in the interval. While the timelocked scripts are very unique and therefore good for identifying channels, and theoretically one could get all LN channels by connection to the LN, this would only be true in a ideal world scenario. So as we will see with our data: $\beta \not\subseteq \alpha$, but keeping the relation in mind will be useful when analyzing the data.

4.1.1 First Interval

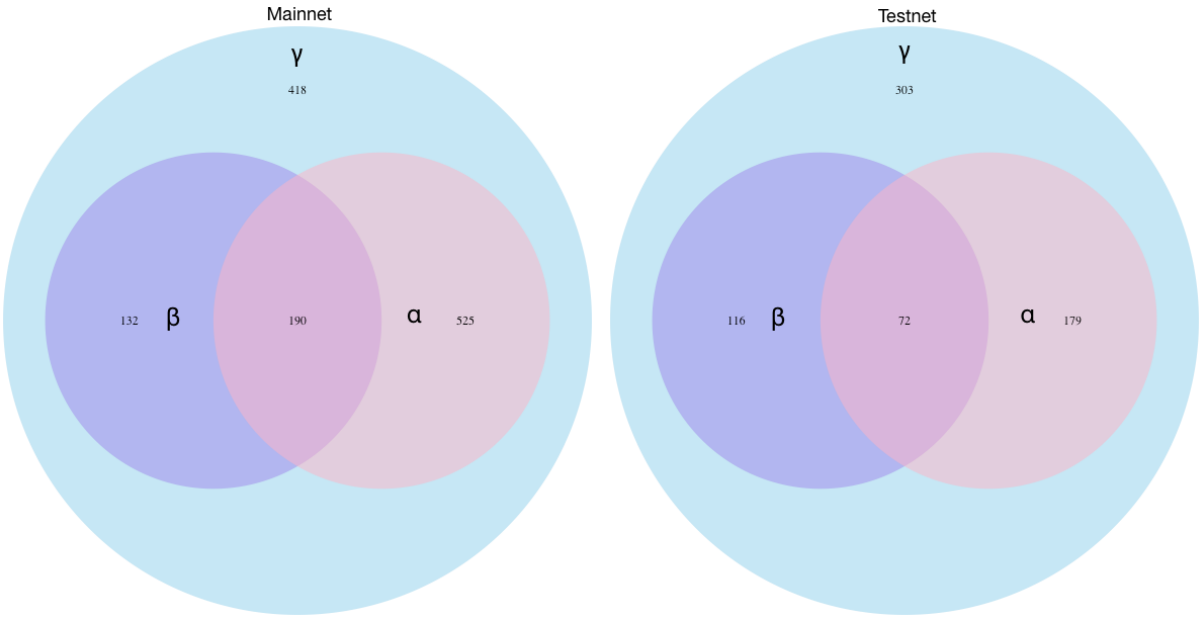

Figure 18: Venn diagram of channel sets in interval one

The first collection interval on the mainnet had a length of 1151 blocks, found between block 517855 and 519005. Our modified LND implementation described in Section 3.2 collected data from the LN and produced the set α containing 715 channels which were closed during this interval. The same interval on the blockchain was parsed two times, once identifying channels using timelocked redeem scripts, and a second time getting all potential channels using multisig identification. This resulted in a β set containing 322 channels and a γ set with 1265 channels shown on the left in Figure 18 as a venn diagram. We can see how both β and α is a subset of γ, but $\beta \not\subseteq \alpha$. The intersection $\beta \cap \alpha$ is the number of channels we identified using the timelocked redeem script which were also found trough the LN node. On the mainnet this intersection was 190 which is 36% of the total channels in α. This is reasonable, as the method will only discover a channel if it has been unilaterally closed as we explained in Section 3.1.2, and not if a channel is closed cooperatively which likely will happen more frequently. Taking the ideal world scenario $\beta \subseteq \alpha$ discussed

above, into account for this data, we have a large $\beta \backslash \alpha$ relative compliment of α in β of 132, which is 40% of the channels in β. Having a false positive this high is very unlikely because the uniqueness of the timelocked redeem scripts, so we made the assumption that these are actually LN channels which we have not been able to capture in our interaction with the LN. The reason we made this assumption was that it is more likely that not every channel is propagated successfully to our single LN node, than there being this many instances of timelocked redeem scripts unrelated to the LN on the blockchain. Addiontally, as we explained in Section 3.2 at least every node running the LND implementation will prune their LN graph of "zombie" channels regularly, so these will eventually no longer be propagated trough the network. This means even if our node does not prune its graph of the network, the channel will never be received in the first place. These are the likely causes of our timelocked channels not being found in the LN channel set. Taking this into account and assuming the union $\alpha \cup \beta$ are all LN channels, the total number of LN channels closed in this interval would be 847. This would entail that 67% of 2of2 multisig transactions in our interval is lightning channels.

For the testnet our LND node collected 838 snapshots starting on block 1290011 and ending on block 1289174. The results are very similar to the mainnet, with the intersection $\beta \cap \alpha$ being 27% of α, and the $\beta \backslash \alpha$ relative compliment of α in β being 62% of channels in β. Again, assuming the union $\alpha \cup \beta$ are all LN channels, 55% of 2of2 multisig transactions is lightning channels. This is however not a upper limit, as our $\alpha \cup \beta$ union is the channels unilaterally closed on the blockchain and channels that have been propagated to our node in the LN. As we discussed our LN node might not be able to get all channels, so there should also be channels not closed unilaterally and therefore not discovered on the blockchain, which we similarly have not been able to collect trough the LN. Which would make the total channel count higher, and more of the 2of2 multisig transactions being channels, than is indicted in our results for both the networks.

4.1.2 Second Interval

For our second interval we only collected data from the mainnet, but tweaking the collection slightly compared to interval one. Instead of only peering with the one assigned when starting the node software we added more, up to a total of 10 peers for the duration of the interval. We also ran the node for a few days before starting the collection, so the node would have opportunity to synchronize as much as possible, and perhaps get channels that would have been pruned in that timeframe and not announced to the node at a later time. A total of 1151 snapshots of the LN was created, giving α 446 closed channels. The timelocked identification on the blockchain found 168 channels for β, while the multisig identification resulted in 667 channels for γ. For this interval the the intersection $\beta \cap \alpha$ was 26% of α, which is similar to the first interval, with 36% and 27% for the mainnet and testnet respectively. However, the $\beta \backslash \alpha$ relative compliment of α in β, was 30% of channels in β, compared to 42% and 63% in the first interval. Also, if the union $\alpha \cup \beta$ all is LN channels, it would entail 75% of the 2of2 multisig transactions are lightning channels. This would indicate we have been able to collect more channels from the LN, which the size of compared to γ

also shows: in this interval α is 67% of γ, while in the first interval this were between 55-56% for both networks. This supports our suggestion made in the previous subsection about there being a number of channels which we are unable to collect trough the LN, so ensuring that the collection node can synchronize as much as possible is important. This would also mean that more than 75% of 2of2 multisig transaction are likely LN channels.

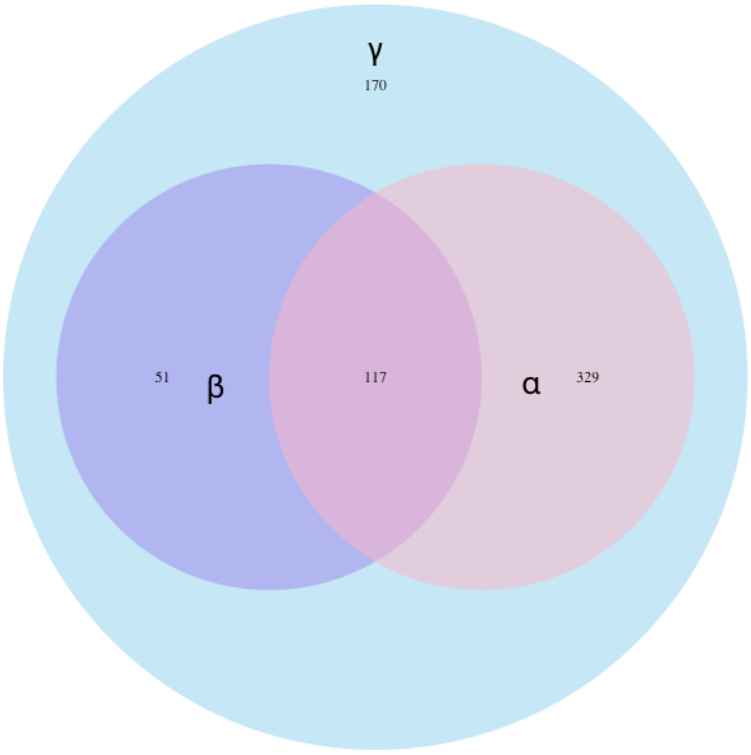

Figure 19: Venn diagram of channel sets during interval two

4.2 Lightning Network size

We mentioned in Section 3.1.4, how we could use the fact that the outputs used for creating a channel has to be of the `P2WSH` type, to determine the maximum possible size of the LN. While parsing the blockchain we have counted all unspent `P2WSH` outputs for each block height, in both the mainnet and the testnet. The results of this can be found in Figure 20, as a graph showing unspent `P2WSH` outputs for different block heights. We can see a distinct difference between the two networks: the mainnet has a more organic growth as a result of real world use, while the testnet graph is characterized by the testing which the network is used for. The sharp decline in unspent outputs on the mainnet at around block 505 000 can be explained by the sharp decline in fees for doing transactions at that time [27], which incentivized users to consolidate their unspent outputs

into fewer bigger ones.

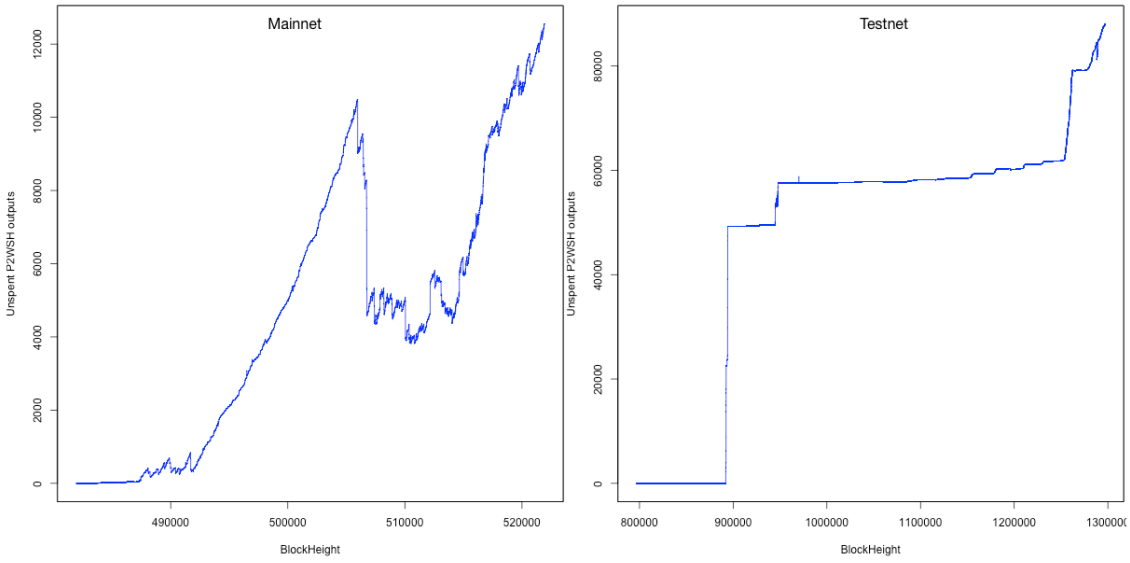

Figure 20: Maximum size of the LN based on unspent P2WSH outputs on the blockchain

While the unspent output count provides a concrete upper limit on the number of channels and therefore size of the LN, we also wanted to know to which degree the amount of unspent `P2WSH` transactions was correlated to the size of the LN. It is clear that changes in the size of the LN will impact the number of `P2WSH` outputs/inputs, but there might be many other transactions not related to the LN having a larger impact. From the data we collected from the LN itself we constructed a total channel count for each block height in the collection interval. Similarly we isolated the same block interval with the `P2WSH` output count from the blockchain. These two variables allowed us to compare the changes in the number of channels in the LN and number of unspent `P2WSH` outputs. We used Kendall rank correlation coefficient on this data for the mainnet, which gave us a correlation coefficient of 0.73 indicating a strong positive correlation. The scatter plot for this can be seen in Figure 21. We did the same for the interval of testnet data we collected, which resulted in a coefficient of 0.45 which is only a moderate correlation. The scatter plot for the testnet can be seen in Figure 22. Both tests has the same p value: $p < 2.2e\text{-}16$. While the two things will be correlated naturally due to the need for channel outputs to be `P2WSH`, the results here indicate there is a strong correlation. The testnet showing a weaker correlation can be due to the unnatural use taking place on the testnet, as was apparent from Figure 20 graph showing unspent output for different block heights. Additionally, the correlation was limited to the amount of data from our collection intervals, meaning we could only correlate the two for around 1000 blocks.

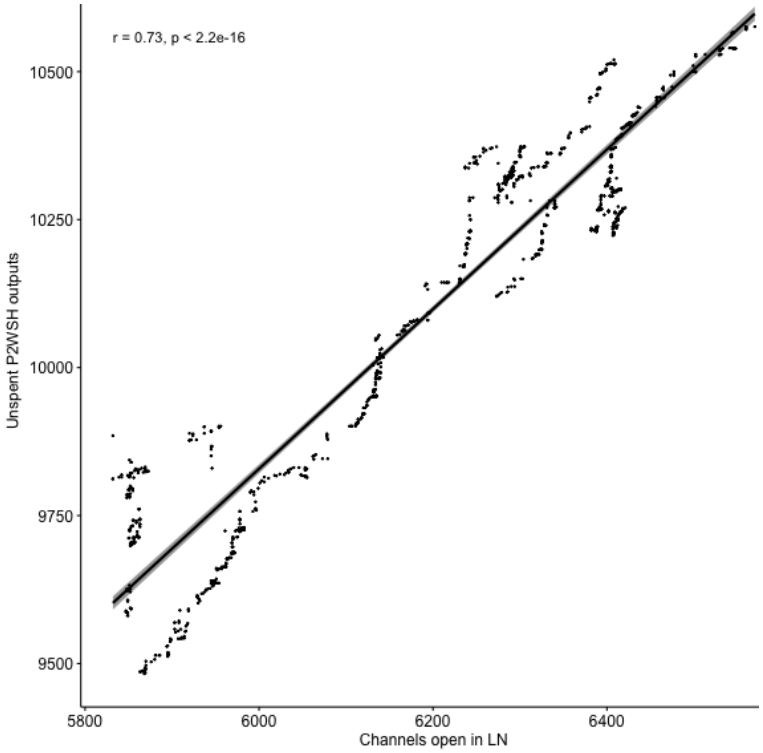

Figure 21: Correlation between unspent P2WSH outputs and channels open in the LN mainnet

4.3 Stats from complete runs

We also tried to run our software with the timelocked redeem scripts method for as long as there was any `P2WSH` transactions on the blockchain, to collect as many channels as possible. As this method is precise and can find a fair amount of channels, it was chosen as the main method for doing this. This resulted in a set of channel graphs for every unilaterally closed channel that has existed up to the point we started parsing. On the mainnet we started on block 524 816 and ended on block 464 816 resulting in a total of 3 809 channels found. For each timelocked redeem script we also checked how the script was evaluated, telling us if the revocation or timelocked clause were used. On the mainnet we found that 163 timelocked outputs were spent with the revocation key, meaning at least this many old commitment transactions was published. In Section 3.1.2 we described how users could spend unconfirmed transactions, resulting in the transactions being found within the same bock. We found a total of 29 such instances, only checking the transactions within our channel graphs. Regarding HTLC frequency on-chain, we found a total of 163 transactions containing HTLC redeem scripts. 101 of these turned out to be spending outputs from closing transactions of graphs we already had identified, which means we could have identified 62 more channels by also using this method. Doing the same for the testnet resulted in 8047 channels found,

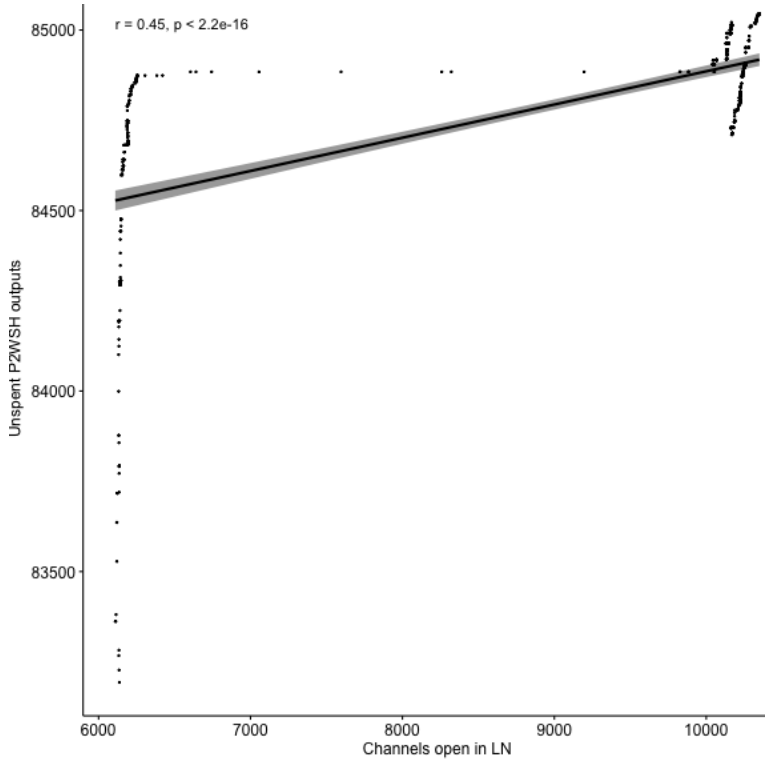

Figure 22: Correlation between unspent P2WSH outputs and LN channels on the testnet

starting on block 1 316 196, and ending on block 816 196. Here we found 38 timelocked outputs spent with the revocation key, and 430 of our transactions being found within the same block. On the testnet we found 566 HTLC transactions, with 143 of them not being connected to any of the channel graphs we had identified. Using these two sets of channel graphs, we extracted stats about the channels as discussed in Section 3.1.4. This gave us data on the value used in each channel as the amount of Bitcoin (btc), and the lifetime of channels in terms of the number of blocks between opening and closing the channel. The results of this can be seen in Table 1. We also included the same stats from the data collected from the LN during our intervals described in Section 4.1. While this can provide some comparison between the stats, as found on the blockchain and from the LN, it will have limits as the blockchain data is for a much longer time interval compared to the LN data. The One clear difference between the mainnet and testnet is the amount of value used for the channels, as we briefly mentioned before coins on the testnet has no value to people, so it would be natural that they are more liberal when creating channels there. We can also see that the testnet data is similar for both the blockchain and LN, while this is not the case for the mainnet. This can indicate that the these stats on the testnet are more constant, while the mainnet channels now has lower value but higher lifetime than the average for all time.

Table 1: Value and lifetime for channels

	Mainnet	LN Mainnet	Testnet	LN testnet
Sample size (channels)	3809	715	8047	251
Average Channel value	0.00439556	0.0023375	0.07086336	0.07758942
Median channel value	0.001	0.0005	0.005	0.06
Standard Deviation Channel value	0.01384113	0.00586706	0.07067473	0.07191958
Average channel lifetime	1 600	1 940	2 441	1 269
Median channel lifetime	693	1 836	433	312
Standard Deviation channel lifetime	2 183	1 791	5 631	4 059

Table 2: Percentages of input - output count for channels

	Mainnet	Testnet
Channels with one input	69.3	76.4
Channels with two inputs	23.2	16.8
Channels with three inputs	4.5	3.8
Channels with more than three inputs	2.8	2.9
Channels with one output	69.2	50.5
Channels with two outputs	29.6	46.3
Channels with three outputs	0.8	2.1
Channels with more than three outputs	0.2	0.8

In Table 2 we see the distribution for the number of inputs and outputs to channels. These results are fairy similar for both networks, we should however note the percent of channels having two outputs in relation to the percent having only one input. For the mainnet the percent of channels having one input is 70%, and the percent of channels only having one output is 71%. As discussed in Section 3.1.4 we cannot easily match input - output pairs to users in the channel, so these numbers either means that many channels does no off-chain transactions at all, meaning the single input will be outputted back to owner, or that all founds in the channel ends up at one of the entities as a result of off-chain transactions. In the testnet column we can see how the single output percentage is lower compared to the single input percent, also the double output percent is higher than the double input. This indicates that many single founded channels ends up with splitting the value when they close, and as we stated in Section 3.1.4 this enables us to say that off-chain transactions has taken place inside the channel. This difference is also present when we checked how many channels had the same number of inputs as outputs. On the mainnet 57% of channels found have the same number of inputs as outputs, while on the testnet this is 46%. Using the HTLC transactions we found which also was outputs from closing transactions in our channel graphs, we can determine how often they occur: 2.6% of channels on the mainnet and 5.2% on the testnet would have one HTLC output. Comparing with the Table 2 we can see how this can make up all of the channels with two outputs, and also the ones with three or more as channels certainly

can have more HTLC outputs.

4.4 Linking

Our software also linked the sets of channels graphs found, using the methods and heuristics described in Section 3.3. The linking was done, both on the set of channel graphs found on the blockchain during the LN collection intervals, and the sets of channels found when doing complete runs as described in Section 4.3. In Table 3 we can see the findings in regards to key reuse, when checking the channels graphs for testnet and mainnet found during the complete runs. Using the 3 809 channels we found on the mainnet, we extracted a total of 26 824 unique keys from the transaction in the graphs, with 1 370 or 5.1% of these being found multiple times. These keys combined where reused 2 850 times, which if the key reuse was evenly distributed in the channel graphs, each channel graph would have 0.75 keys used in another channel graph. For the 8 047 channels found on the testnet, we got 50 050 unique keys. The results here are very similar to the mainnet, with 6.5% of the keys found were reused, and 0.9 reused keys per channel graph if they were equally distributed. As discussed in Section 3.3 regarding key reuse: key reuse within the same channel graph does not let us allow to link channels, but our results below does not take this into account, but we will address it later in this section.

Table 3: Key reuse in channel graphs found in complete runs

	Mainnet	**Testnet**
Unique Keys found	26 824	50 050
Unique keys reused	1 370	3 250
Instances of key reuse	2 850	7 240
Maximum number of reuses for a single key	23	171
Average reuse of keys	2	2.2

Our two other heuristics was also used on these channel sets, allowing us to determine how many channels we were able to link using these. The results of this is shown in Table 4. For heuristic one, where we found directly connected graphs trough outputs - inputs, we have separate results for each possible output which can be the source of such a relation. We discovered a clear difference between the different outputs: the outputs from the founding transactions makes up over 65% of the connections found using heuristic one, both on the testnet and mainnet, compared to 2.1% or less for the outputs from the closing transactions. The outputs from the timelocked transactions makes up 30% and 33% of the connections on the mainent and testnet respectively. One possible explanation for this difference has to do with the value of the outputs: closing and timelocked outputs are used to give one party their value from the channel, meaning the value of these outputs is limited to the channel balance of the participant owning them; outputs from the founding transaction on the other hand, is used to transfer change resulting from the output spent to found the

channel, which has no size limit, meaning a output many times the value of the channel can be used. Unless multiple inputs is used to found a channel, creating a new channel using the outputs from another will limit the value of the founding input to the balance in the previous channel. A possible reason for the difference between closing and timelocked outputs could be the recommendation to spend timelocked outputs discussed in Section 3.3, which means if one is closing many channels unilaterally at the same time, consolidation of all timelocked outputs into one timelocked transaction is a natural thing to do; this would cause graph overlap as defined in heuristic three, but also create a high value output from the timelocked transaction which could be used to found new channels. Heuristic 3 which used graph overlap or multi-input timelocked transactions provided 12% on mainnet, and 24% on the testnet, of the total connections found using both heuristic two and three.

Table 4: Relations found between channel graphs using heuristic one and two

	Mainnet	**Testnet**
Total relations found	1586	5 715
Founding output relations	931	2 824
Closing output relations	30	2
Timelocked output realtions	435	1 499
Graph overlap relations	190	1 390

If we combine our results of using all three heuristics on the channel sets, we get a total of 4 436 relations between the channel graphs in the mainnet. This enabled us to crate 2 182 links between channels. The reason this this difference is redundancy for the connections, and key reuse within a single channel graph. This means that 2254 of our relations, or about 50% where redundant, but only 36 of them being reuse of the same key in the same channel graph. The set of linked channels can be seen visualized in Figure 23, were we have created a channel network as discussed in Section 3.3. The network consists of 2 324 smaller components, with each node having a average degree of 1.1. While this graph does not distinguish users, we can based on how these components is structured reveal additional information about the users participating in the channels. As edges represent at least one common user being present both nodes, we can create the property: for adjacent nodes there can be at most three different users, and minimum two users. This means if the distance between vertices is more than one, we can no longer be sure if if there is common user participation, but we cannot rule it out either. More importantly, it has a impact on complete components-i.e., every node is connected to every other node. Because the property we defined must hold for every node in such a complete component, there are only two possibilities with regard to user participation: there can be at most three total users in the component, or that one user participates in all nodes, but with a different second user in each. So either there are three total users in such a component, or the number of nodes plus one total users, because each node has a different user plus the one participating in all. We can see a few examples of such components in Figure 23, and also some components having a similar structure, but having some additional nodes

which is not part of the complete core of the component.

We did the same as discussed in the previous paragraph for the set of channel graphs from the testnet. The total relations using all three herusitcs was 12 955, while the amount of links we could create was 7084. Again 45% of the connections we found where redundant for actually linking the channels. Here the number of reused keys within a single channel graph was 113, which is relatively small compared to the 5758 other redundant relations. The channel network shown in Figure 24 is more connected than the mainnet in Figure 23, with the testnet network having over twice as many nodes, but only having 617 more components. We can see this difference clearly with the large component in the middle of the testnet graph. As the LN is one network, it should consist of one dynamic component, meaning a historic network graph should also ideally have one large component. While we can see this beginning to emerge in our testnet channel network, it is very clear in the channel network constructed using LN data as seen in Figure 25.

The channel network in Figure 25 based on LN data from interval 1, was created so we could compare it with the networks we crated by linking blockchain information. This would allow us to see how connected a channel network should ideally be if all relations where available to us in the blockchain. In the LN data we collected during the intervals we used the node id, which identifies a node/user, to link the channels that nodes participated in. While users are easily separable by this id, we ignored this to create the same type of network as we did using the blockchain data, meaning edges only indicate common user participation, not a specified user. As shown in Figure 25 the network is highly connected with the 715 nodes having a average degree of 58. We can see that almost all nodes are a part of the main component, with only having 6 nodes not being connected to it. Figure 32 shows the histogram for the node degrees in this network. In this histogram there is one node that sticks out, it has a degree of over 200, compared to the other highly connected nodes which have a degree of around 150. There is also a high frequency of nodes having a degree between 100 and 150. In Table 5 we have taken the top five nodes in terms of degree, and calculated some centrality measures for them. The highest connected node with a degree of 239, has also the highest score in the other measures, it can be seen in Figure 25 as the red node. For the other nodes there is some variation in the scores, but all five nodes being in the top ten for all nodes in every measure, except node or channel 569069835159470080 colored blue in Figure 25. For closeness centrality this node was ranked 31, while scoring high on degree and betweenness centrality. Comparing this network to the channel network in Figure 30, which we cratered using the channels identified on the blockchain in the same interval, and linked using our three heuristics, we can see how limited our linking capabilities really are. Having almost six times as many nodes as edges and a average degree of 0.1, we can see how most channels identified has no relation to others we could find, within such a short interval. The same networks for the testnet interval and the second interval for mainnet can be found in the Appendix A.1

Table 5: Centrality measures for the highest degree nodes for the network in Figure 25

(short) Channel Id	Degree centrality	Closeness centrality	Betweenness centrality
565863659278368768	239	0.0001790831	23084.374
565758106242187265	176	0.0001743679	10285.468
569069835159470080	159	0.0001737619	11597.068
566469490153553920	153	0.0001763668	5733.209
567496434090377216	152	0.0001756543	6881.286

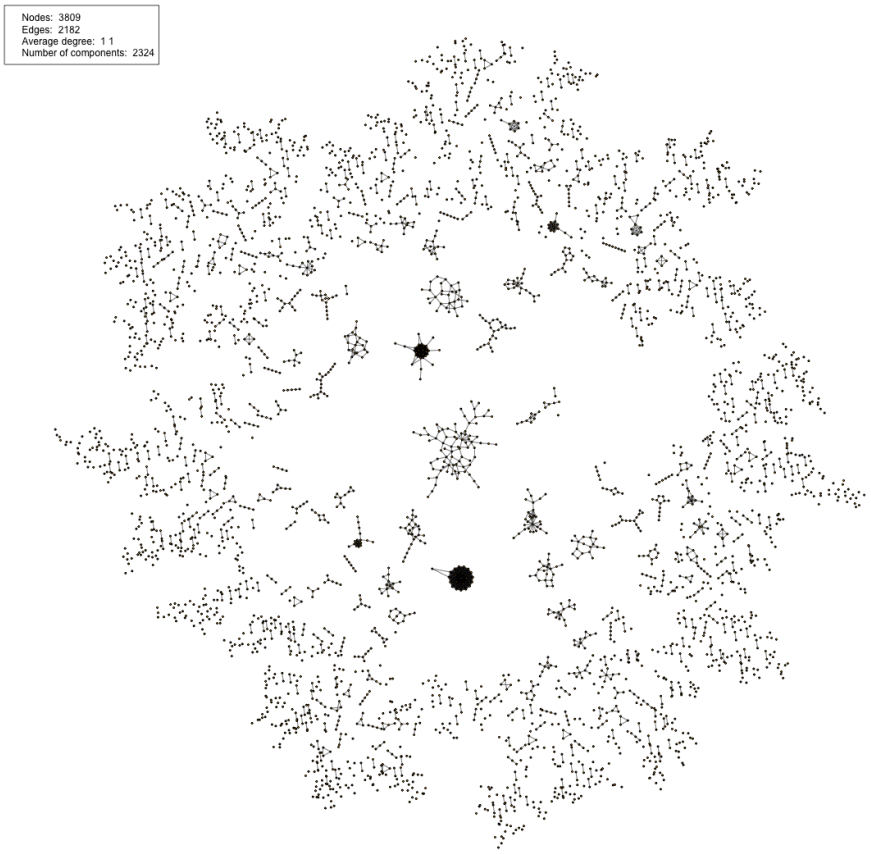

Figure 23: Channel network for mainnet

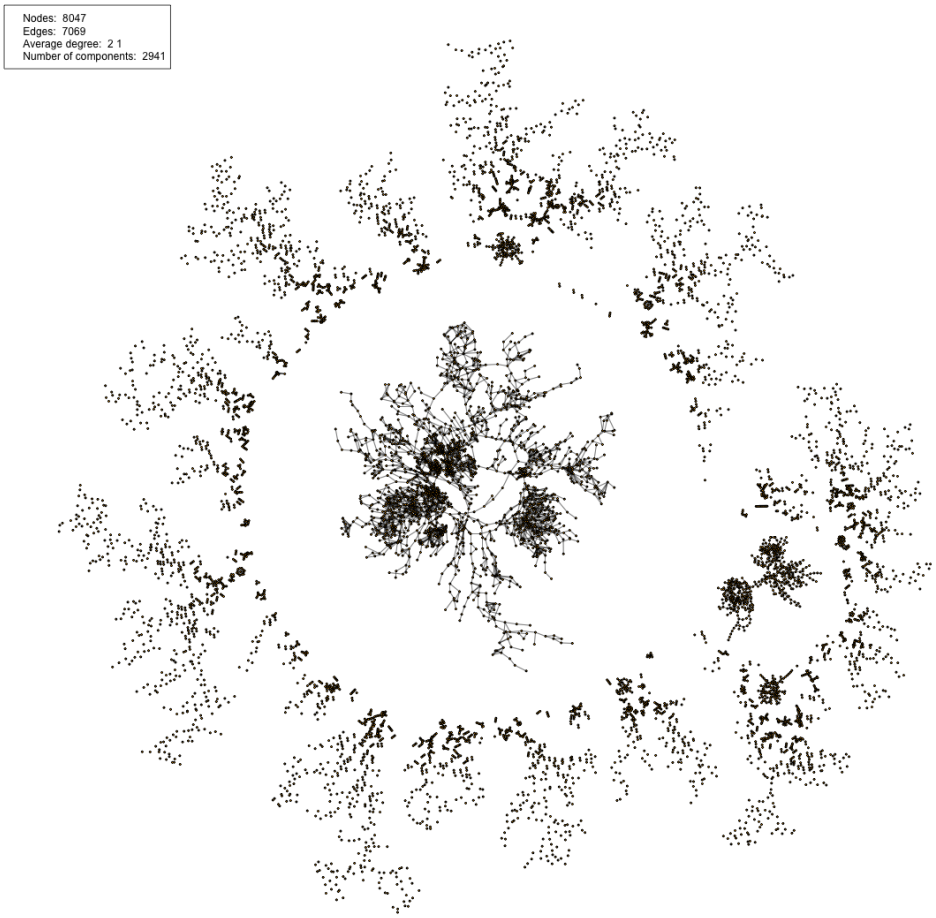

Figure 24: Channel network for testnet

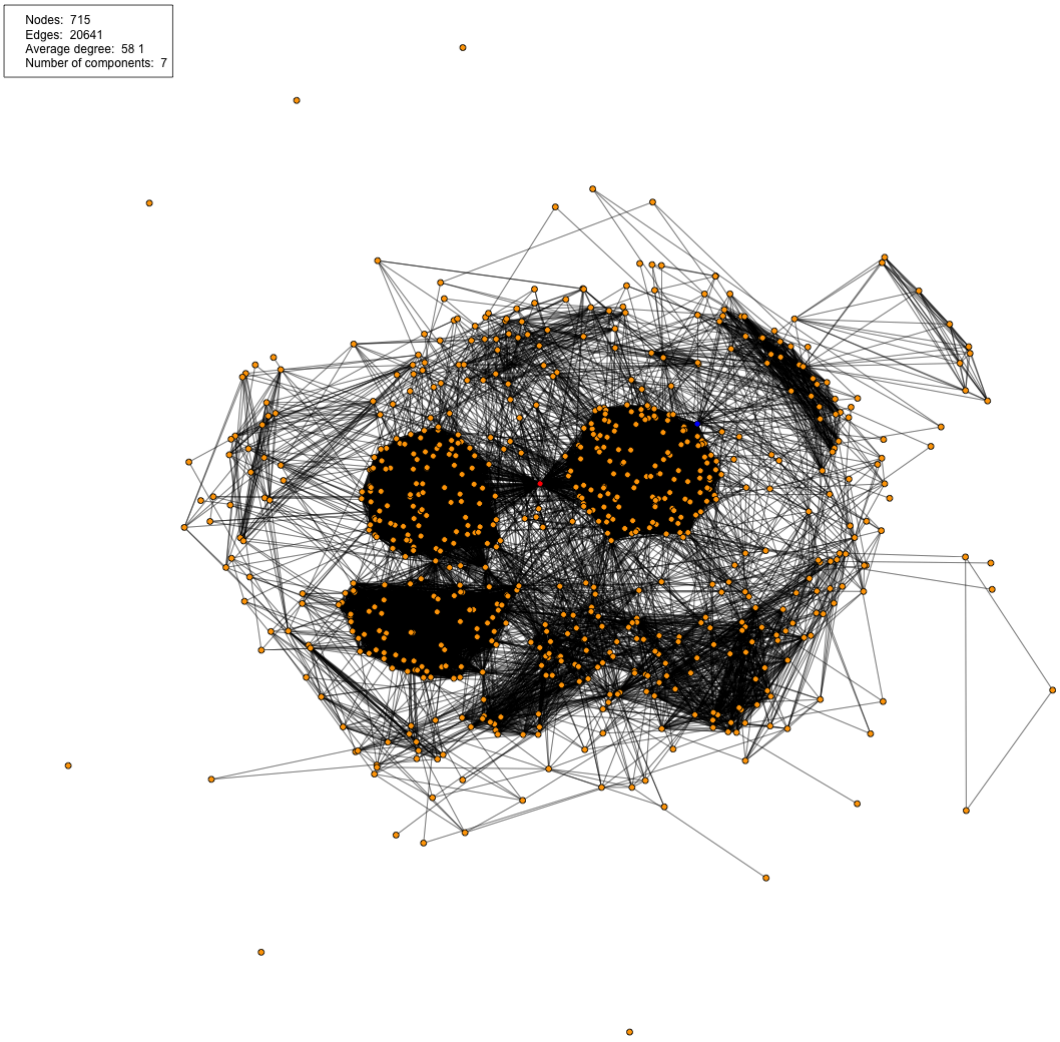

Figure 25: Channel network from the LN mainnet, first interval

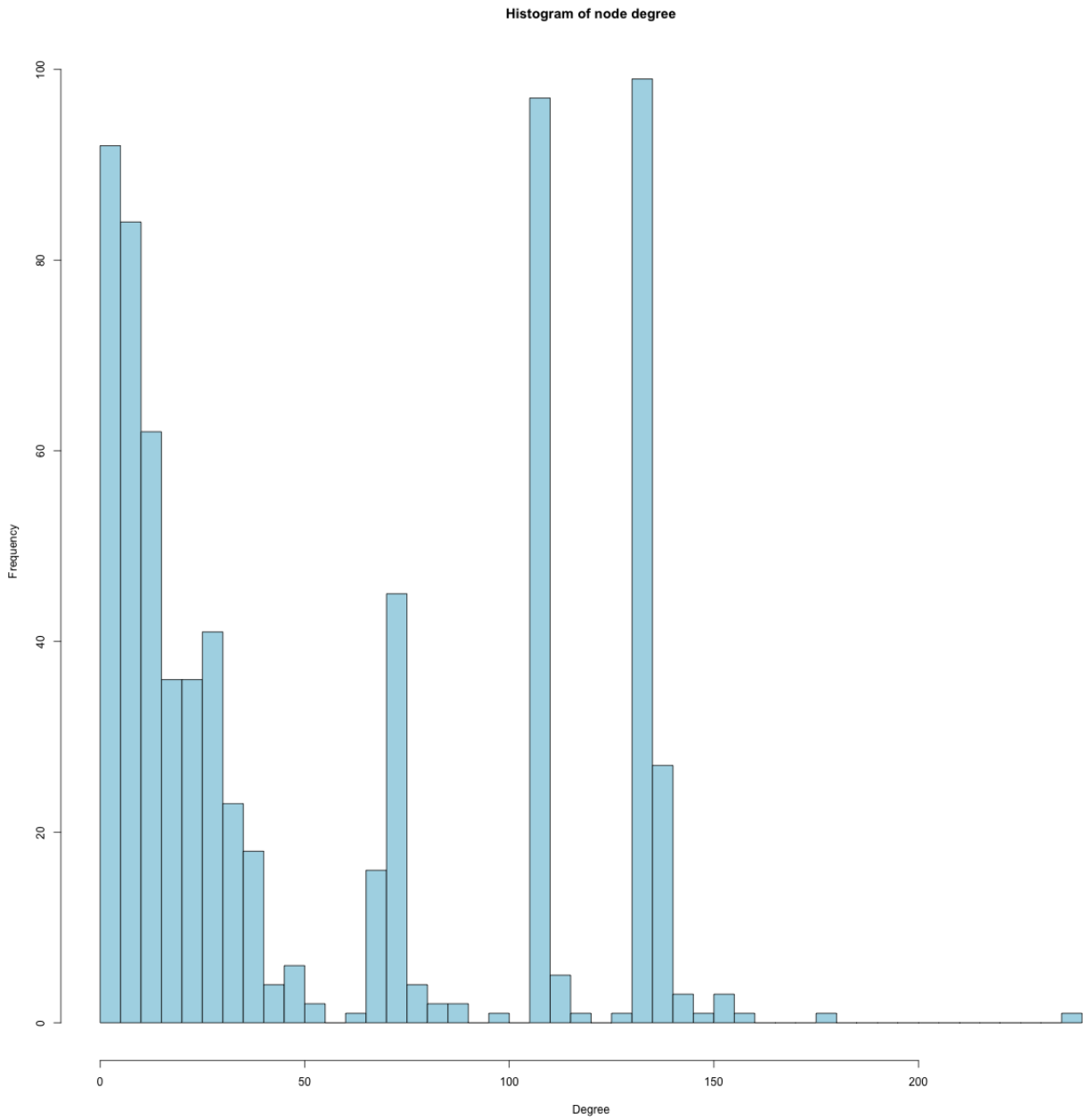

Figure 26: Histogram of node degree for the network in Figure 25

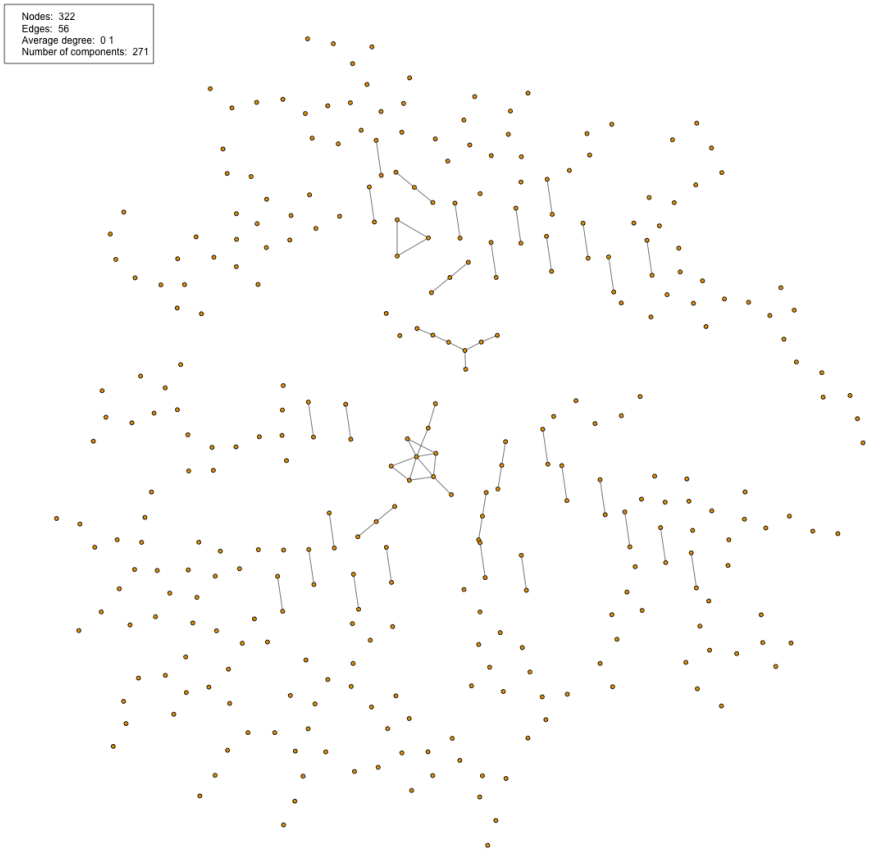

Figure 27: Channel network from the blockchain, mainnet, first interval

5 Summary

5.1 Conclusion

In this project we have explored the LN from the blockchain perspective. As the LN does not change the Bitcoin system in any way, the older analysis methods are still valid if we don't consider the LN. Taking the LN into consideration means seeing transactions in the context of the LN in addition to the context of the blockchain. As we have discovered when attempting to locate on-chain LN transactions: one can never be perfectly sure that a transaction is LN related or not, if we only use the informatin available in the blockcahin. The different characteristics used to identify potential on-chain transactions have different level of uniqueness, but will limit the amount we can identify. Uniqueness makes it very unlikely that the transaction is not LN related, but as stated previously: users are free to create whatever scripts and transaction chains they wish. The identified channels does also contain limited amount of information allowing us to infer things about the LN, especially in terms of off-chain transaction. As the on-chain transactions facilities the channels in the LN, most of the information we get are related to these and its users. While this information might allow for us to determine some of the structure of the LN, in terms of which channels are connected, it does not in our opinion help in determining anything about major regarding off-chain transaction flows. We can therefore conclude, from the blockchain analysis perspective, that doing transfers within the LN provides much better privacy than doing transfers on-chain.

Our results also show some possibilities. In our second interval comparing the channels found with our two methods of identification, and channels collected trough the LN, we see that possibly more than 75% of all 2of2 multisig transactions is in fact LN channels. This was due to the possible limits of our LN node to collect all channels, as other nodes will remove inactive channels, and they will therefore not be propagated. This might become apparent if one where to use longer collection intervals, or run the collection node longer before starting to collect, allowing the node to receive many channels that will eventually be pruned by other nodes. The timelocked identification provides a good middle ground between precision and how many channels it will be able to find, with the results from the second interval indicating that around 25% of channels is unilaterally closed. As shown in our correlation between `P2WSH` outputs and number of channels on the LN, the `P2WSH` outputs can be used as an indicator of the number of channels, at least providing a upper limit on active channels at any point in time. It is important to note however that these things are very reliant on the current use cases of the things used for identification and correlation. For example, if 2of2 multisig is used commonly for other porposes than LN channels, identification based on it will become extremely unreliable.

We should take note that much of established principles for linking information on the blockchain is still relevant for this case. Using new keys for each transaction providing the users with pseudoanonymity is very much still important for providing privacy in channels graphs. In addition to enabling us to link channels containing the same keys, it will also enable linking of keys within the channel graph itself, as keys are related to themselves. We discussed previously how differentiating users within a graph would enable us to determine which users provide link between channels, allowing us to recreate parts of the structure of the LN more accurately. The best way of doing this would be to link all the keys, and then the entire transaction graph should also be used. Our linking results show that the information available and our heuristics are very limited. When comparing the channel network created by using information from the LN, with the one created using channel graphs found in the blockchain in the same interval, we can see a very clear difference in the number of edges. The network using LN information has a average degree over a hundred times larger than the channel network from the blockchain. The effectiveness of our heuristics is dependent on user behaviour, and how LN implementations operates in regard to transaction structure. For example, using a single transaction to spend many timelocked outputs is simple and convenient, but allows for linking using our third heuristic.

It is apparent that if one wants information about the LN, it is in most cases better to simply get this directly from the LN, instead of finding it on the blockchain. But as we previously have pointed out: all LN channels needs to be anchored on the blockchain, which contains verified historical data, so it will be a important source of information with regards to the LN. Additionally, blockchain analysis can be used in conjunction with other methods of analysis on the LN, providing a supplement.

5.2 Future Work

As a emerging technology the LN and Bitcoin system is being changed and improved regularly, and because the work presented here is for the current state of these systems, any future work using elements from this project, will have to change methods and assumptions. A example is founding transactions: currently founding transactions can be used to found a single channel, but there is nothing in the way of having a single founding transaction for many channels [28]. This would provide additional possibilities for linking. It could also be used for identification of channels-e.g., one channel might be identified because of a timelocked output, and by finding ouputs of other channels in the founding transaction we can also identify them.

Using HTLC transactions to identify on-chain LN transactions are something that could be explored further. In Section 4.3 we presented our result of checking their frequency, and overlap with already found channel graph. Using such a identification method would have given us 62 more channels for the mainnet and 143 for the testnet. Nevertheless, the extra transactions included in a channel graph can provide more opportunities for linking. The scrips for HTLC transactions should also be explored further, to determine the value of the data within them. The preimage `R` is unique

for one off-chain transaction, making it a possible element to use for getting information about off-chain transactions.

One of the proposed improvements that will impact this type of work the most is channel factories [29]. The proposition is to have a additional layer between the blockchain and the payment channel network, with the purpose of creating the one-to-one payment channels which makes up the payment channel network. The channel factories on this layer can manage channels, meaning payment channels no longer requires any on-chain transactions to manage. The channel factories itself requires on-chain transactions, but with such a additional layer of abstraction, allowing for many dynamic channels per on-chain transactions, the use and methods of blockchain analysis must be reassessed.

Bibliography

[1] Poon, J. & Dryja, T. 2016. The bitcoin lightning network: Scalable off-chain instant payments. 2016. *URl: https://lightning.network/lightning-network-paper.pdf (visited on 2018-06-01)*.

[2] Nakamoto, S. 2008. Bitcoin: A peer-to-peer electronic cash system.

[3] Trillo, M. 2013. 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 2018-05-28)*.

[4] Reid, F. & Harrigan, M. 2013. An analysis of anonymity in the bitcoin system. In *Security and privacy in social networks*, 197–223. Springer.

[5] Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G. M., & Savage, S. 2013. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, 127–140. ACM.

[6] Herrera-Joancomartí, J. 2015. Research and challenges on bitcoin anonymity. In *Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance*, 3–16. Springer.

[7] Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M., & Ravi, S. 2017. Concurrency and privacy with payment-channel networks.

[8] Antonopoulos, A. M. 2017. *Mastering Bitcoin: Programming the Open Blockchain*. " O'Reilly Media, Inc.".

[9] Andresen, G. 2012. Bitcoin improvement proposal 16. *URl: https://github.com/bitcoin/bips/blob/master/bip-0016.mediawiki (visited on 2018-03-28)*.

[10] Lombrozo, E., Lau, J., & Wuille, P. 2015. Bitcoin improvement proposal 141. *URl: https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki (visited on 2018-04-02)*.

[11] Lau, J. & Wuille, P. 2016. Bitcoin improvement proposal 143. *URl: https://github.com/bitcoin/bips/blob/master/bip-0143.mediawiki (visited on 2018-04-02)*.

[12] Wuille, P. 2014. Bitcoin improvement proposal 62. *URl: https://github.com/bitcoin/bips/blob/master/bip-0062.mediawiki (visited on 2018-06-01)*.

[13] Lightning Network Specifications. 2018. Bolt 3: Bitcoin transaction and script formats. *URl:* https://github.com/lightningnetwork/lightning-rfc/blob/master/03-transactions.md *(visited on 2018-04-24)*.

[14] Pfitzmann, A. & Köhntopp, M. 2001. Anonymity, unobservability, and pseudonymity—a proposal for terminology. In *Designing privacy enhancing technologies*, 1–9. Springer.

[15] Androulaki, E., Karame, G. O., Roeschlin, M., Scherer, T., & Capkun, S. 2013. Evaluating user privacy in bitcoin. In *International Conference on Financial Cryptography and Data Security*, 34–51. Springer.

[16] Ron, D. & Shamir, A. 2013. Quantitative analysis of the full bitcoin transaction graph. In *International Conference on Financial Cryptography and Data Security*, 6–24. Springer.

[17] Herrera-Joancomartí, J. & Pérez-Solà, C. 2016. Privacy in bitcoin transactions: new challenges from blockchain scalability solutions. In *Modeling Decisions for Artificial Intelligence*, 26–44. Springer.

[18] blockchain.info. 2018. Blockchain size. *URl:* https://blockchain.info/charts/blocks-size?timespan=all *(visited on 2018-04-23)*.

[19] Russell, R. 2015. Routing on the lightning network? *URl:* https://lists.linuxfoundation.org/pipermail/lightning-dev/2015-July/000019.html *(visited on 2018-06-01)*.

[20] roasbeef. 2018. Btcd - an alternative full node bitcoin implementation written in go (golang). *URl:* https://github.com/roasbeef/btcd *(visited on 2018-04-26)*.

[21] Bitcoin Wiki. 2018. Script. *URl:* https://en.bitcoin.it/wiki/Script#Script_examples *(visited on 2018-04-29)*.

[22] BtcDrak, Friedenbach, M., & Lombrozo, E. 2015. Bitcoin improvement proposal 112. *URl:* https://github.com/bitcoin/bips/blob/master/bip-0112.mediawiki *(visited on 2018-04-26)*.

[23] Lightning Network Specifications. 2018. Lightning network in-progress specifications. *URl:* https://github.com/lightningnetwork/lightning-rfc *(visited on 2018-04-26)*.

[24] lightningnetwork. 2018. Lightning network daemon. *URl:* https://github.com/lightningnetwork/lnd *(visited on 2018-05-0)*.

[25] Lightning Network Specifications. 2018. Bolt 7: P2p node and channel discovery. *URl:* https://github.com/lightningnetwork/lightning-rfc/blob/master/07-routing-gossip.md *(visited on 2018-05-22)*.

[26] Lightning Network Specifications. 2018. Bolt 5: Recommendations for on-chain transaction handling. *URl: https://github.com/lightningnetwork/lightning-rfc/blob/master/05-onchain.md (visited on 2018-04-24)*.

[27] Hoenicke, J. 2018. Johoe's bitcoin mempool statistics. *URl: https://jochen-hoenicke.de/queue/#1,all (visited on 2018-05-11)*.

[28] Decker, C. 2017. [lightning-dev] is it possible to fund multiple channels using single transaction with multiple output? *URl: https://lists.linuxfoundation.org/pipermail/lightning-dev/2017-December/000869.html (visited on 2018-05-28)*.

[29] Burchert, C., Decker, C., & Wattenhofer, R. 2017. Scalable funding of bitcoin micropayment channel networks. In *International Symposium on Stabilization, Safety, and Security of Distributed Systems*, 361–377. Springer.

A Appendix

A.1 Additional channel networks

A.1.1 Second interval mainnet

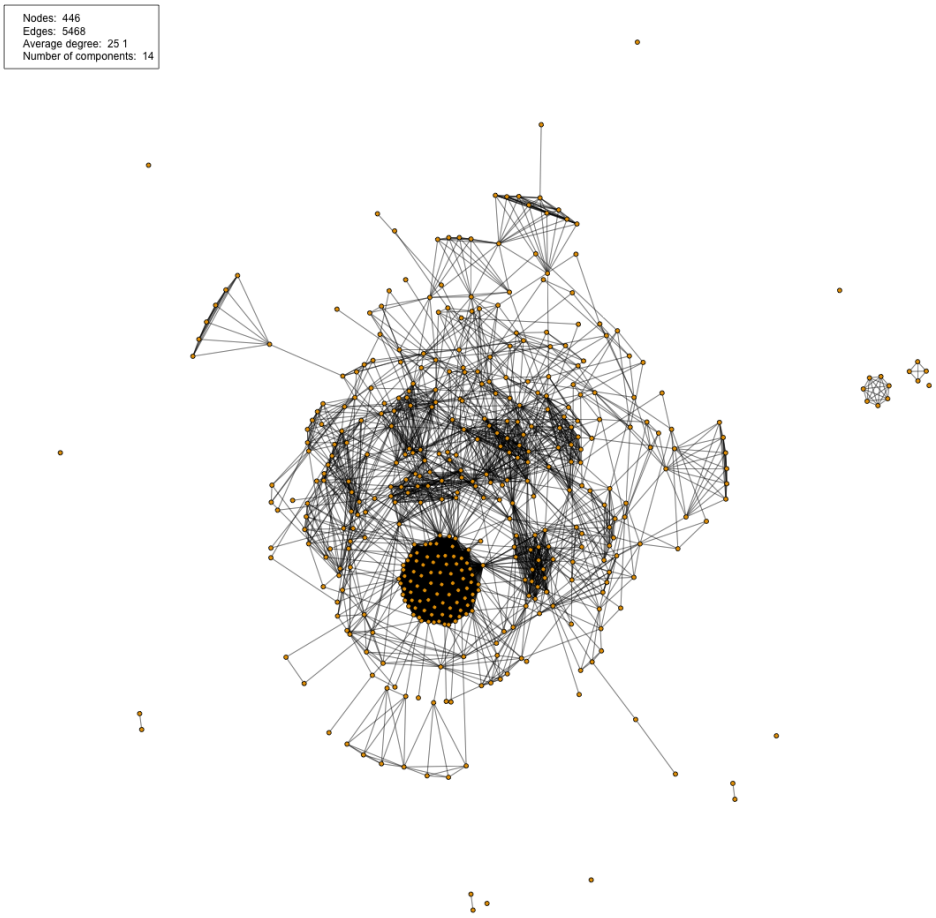

Figure 28: Channel network from the LN, mainnet, second interval

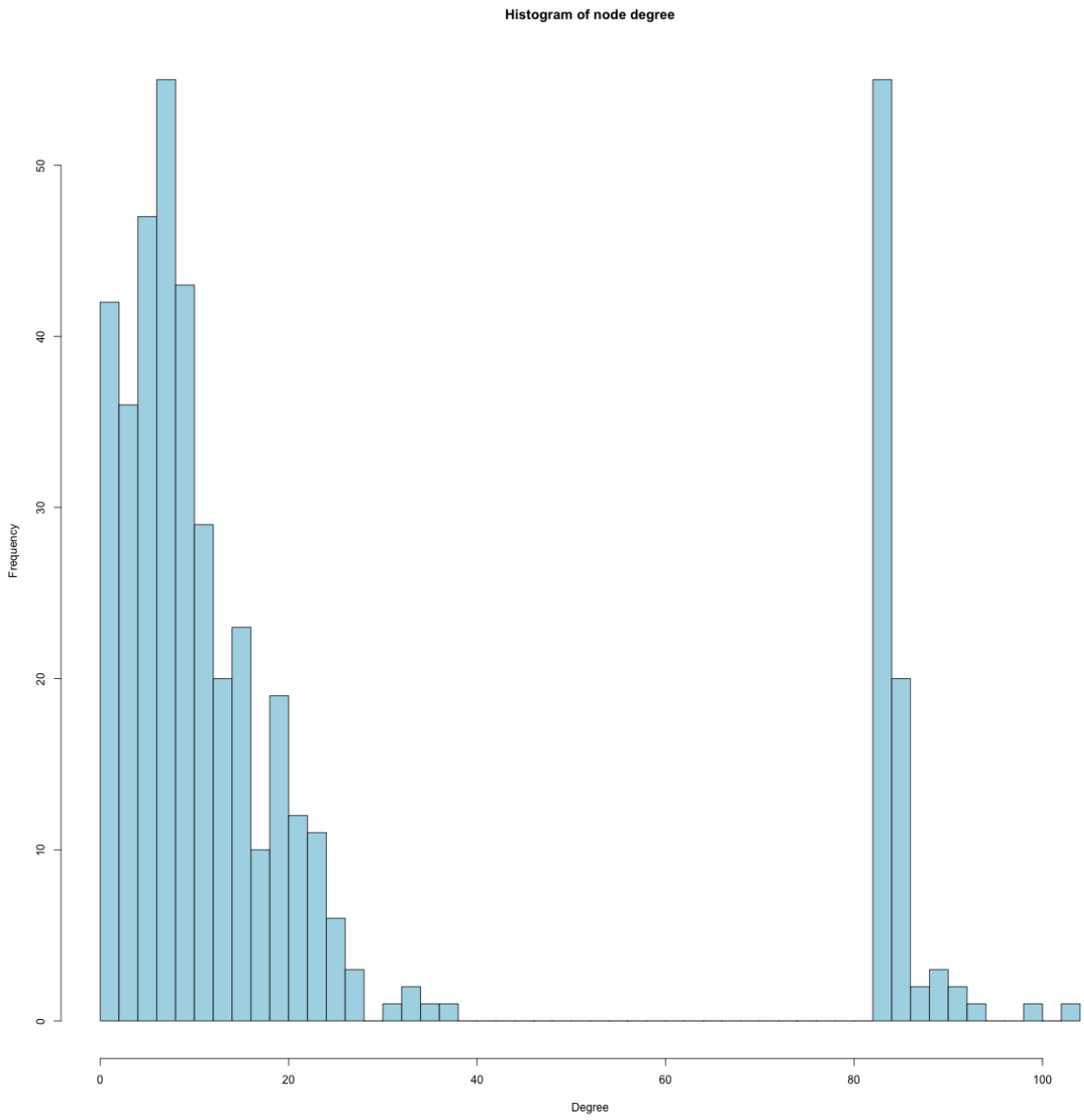

Figure 29: Histogram of node degree for the network in Figure 28

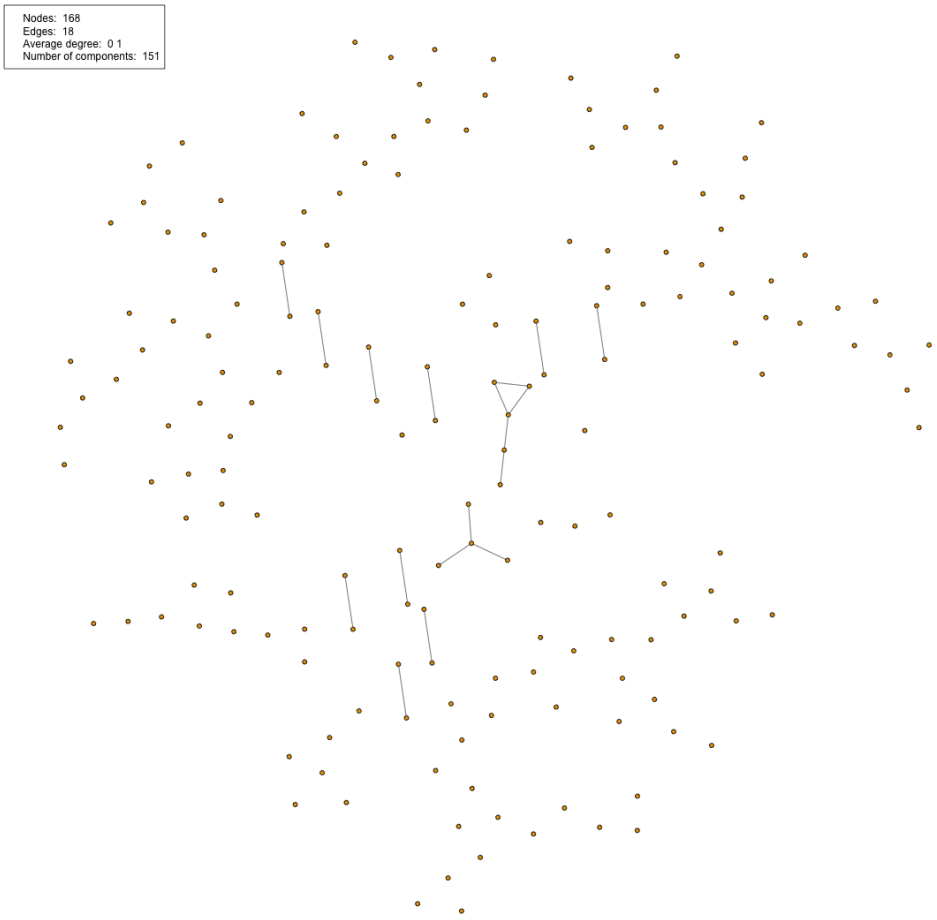

Figure 30: Channel network from the blockchain, mainnet, second interval

A.1.2 Testnet

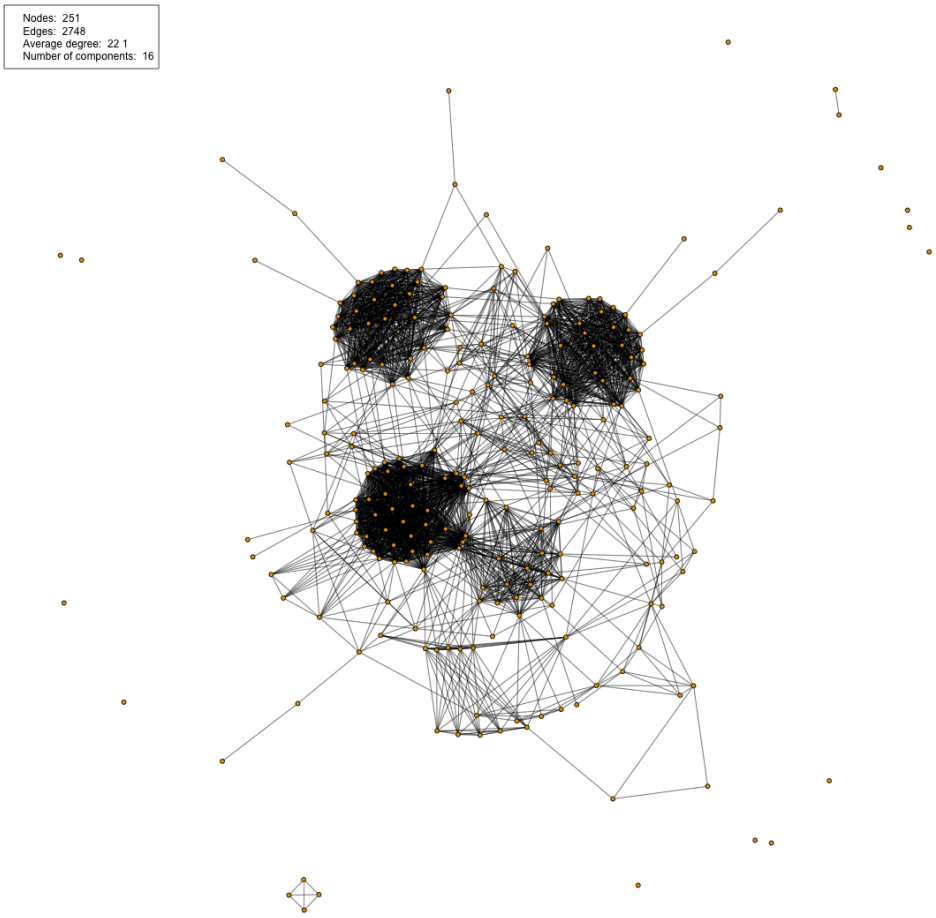

Figure 31: Channel network from the LN testnet, from interval one

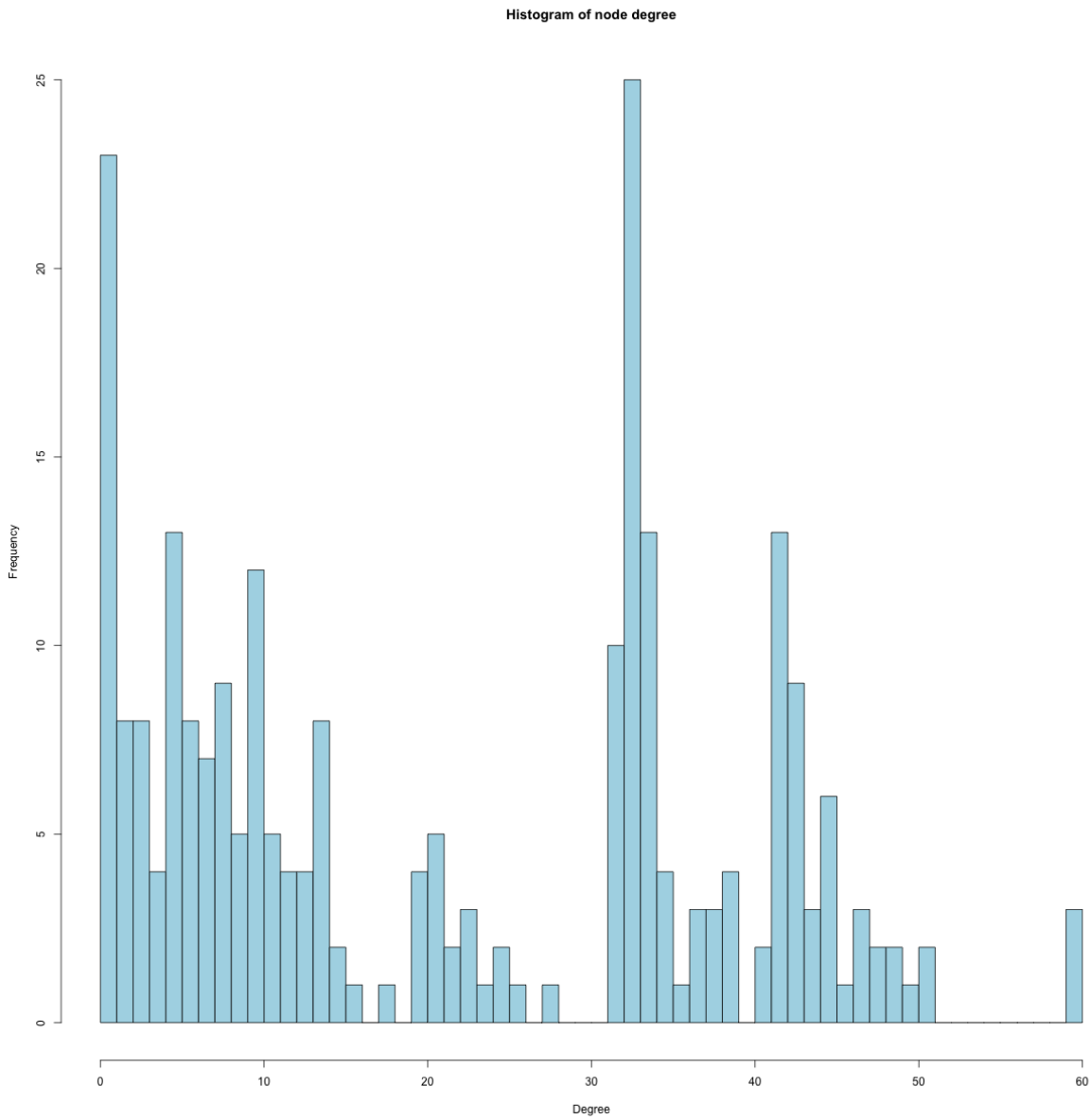

Figure 32: Histogram of node degree for the network in Figure 31

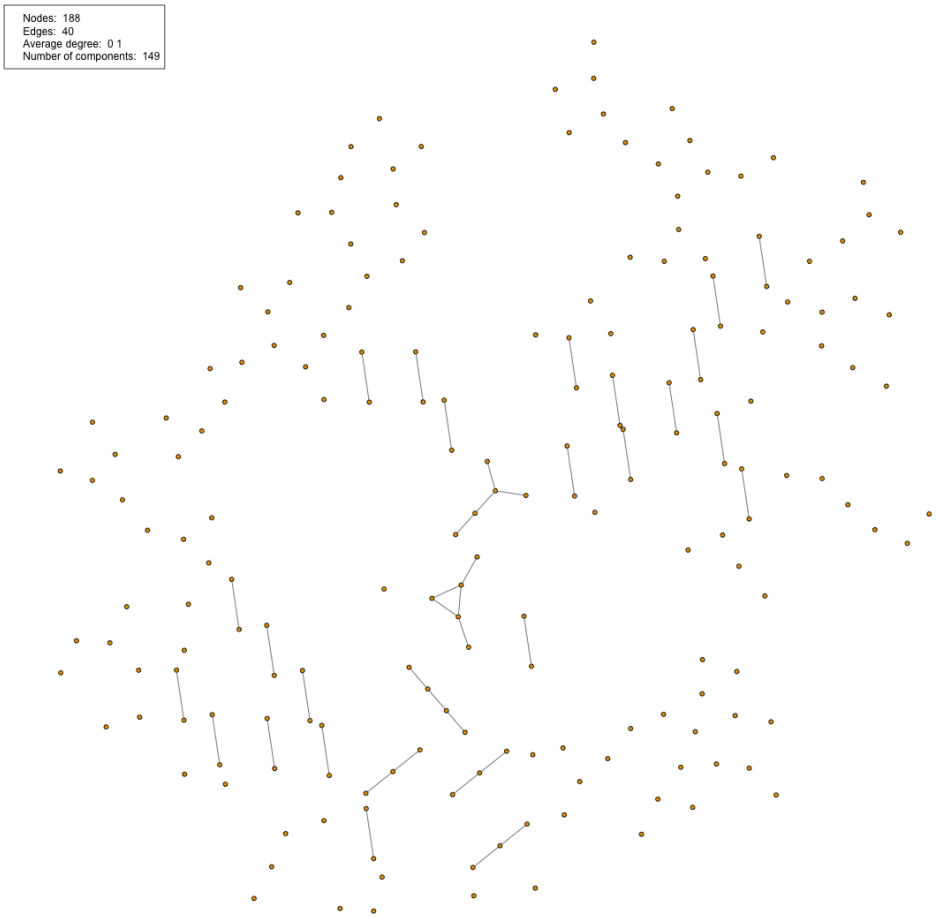

Figure 33: Channel network from the testnet blockchain, from interval one