Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Routing Scalable Bitcoin Payments

Bachelor Thesis

Adrian van Schie

vadrian@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Christian Decker
Prof. Dr. Roger Wattenhofer

September 18, 2015

Abstract

With current block size limitations, Bitcoin does not scale to a level where it can be used in everyday life by the whole world. As remedy, off-blockchain solutions have been proposed. Bidirectional payment channels allow users to make secure payments with instant confirmation, unlike blockchain transactions whose confirmation takes several minutes. A network of payment service providers, connected by bidirectional micropayment channels, enables clients to make payments to each other that are routed over the network. In this thesis we propose a routing protocol that provides routing information exchange and payment routing.

Contents

Chapter 1

Introduction

Bitcoin [4] has several advantages over centralized payment systems. Due to its high decentralization, Bitcoin is more resilient; there is no central point of failure. With the blockchain, a transaction database shared by all nodes on the Bitcoin network, all transactions are publicly available enabling anyone to verify transactions. Despite this transparency, the identity of people behind payments is private by default.

One prevailing disadvantage of Bitcoin to other payment systems is scalability. Bitcoin can handle a maximum of 7 transactions per second with the current maximum block size of 1 MB. VISA, on the other hand, handles on average 2000 transactions per second. Bitcoin can only match or even surpass this number if the scalability problem has been resolved.

The main issue is that every Bitcoin transaction is stored on the blockchain. One proposed solution to support more transactions is increasing the block sizes. Larger blocks, though, make full nodes more expensive to operate: more storage is needed, higher bandwidths are required. Higher operating costs have a centralizing effect, since running a full node becomes more expensive. One of Bitcoins core features, though, is its decentralized nature. Fewer entities running full nodes with higher operating costs could also result in higher transaction fees, since they want to remain profitable. Higher fees may keep users from making small payments, hurting the development of Bitcoin.

Another approach to tackle the scalability problem are off-blockchain solutions, where only a few Bitcoin transactions need to be committed to the blockchain. These solutions are still trustless and secure. Additionally, they grant instant confirmation of transactions and don't take minutes to be confirmed as is the case with current Bitcoin transactions on the blockchain. The foundation of such off-blockchain solutions are micropayment channels that allow two parties to securely make payments to each other without needing a commit to the blockchain for each transfer. These schemes summarize the transfers made between the parties of the micropayment channel in a single Bitcoin transaction.

A network of client nodes and payment service provider (PSP) nodes, con-

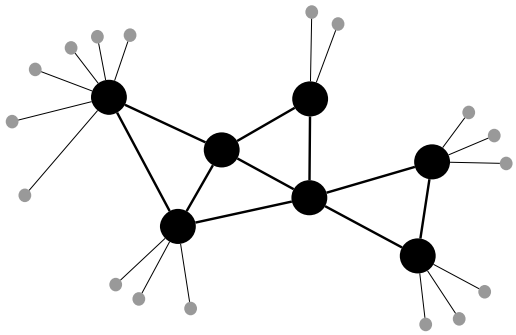


Figure 1.1: A payment network consisting of clients nodes (grey) and payment servicer providers (black).

nected by bidirectional micropayment channels as in figure 1.1, represents a scalable solution. Clients are able to make payments among each other, that are routed over PSP nodes, with guaranteed end-to-end security.

In this paper we propose a routing protocol for a network of bidirectional micropayment channels. The protocol manages routing information exchange as well as payment routing over the network. Routing information exchange is performed in a similar way to the Border Gateway Protocol (BGP) for the Internet. The major difference, though, is that the connections in our network are limited by capacities: only a certain amount can be maximally sent to another node. In addition, the fact that payments can be split up opens new possibilities for routing decisions.

We implemented the protocol in a modular fashion in order to reuse it for real-world scenarios as well as simulations. We achieved high payment success rates in simulations with our proposed protocol.

1.1 Related Work

A faster block generation rate can be achieved with the GHOST protocol proposed by Sompolinsky and Zohar [7] by modifying the selection of the main chain. Fast payments were shown to be susceptible to double-spend attacks [2, 8].

Contracts and Simple micropayment channels have been introduced by Hearn [3]. Work on Hub-and-Spoke Micropayments has been done by Todd [9]. The Lightning Network by Poon and Dryja [5] comprises a network of bidirectional payment channels. Duplex Micropayment Channels introduced by Decker and Wattenhofer [1] are bidirectional channels that allow the parties to reset a channel, e.g., when one direction of the channel is depleted.

The Border Gateway Protocol [6] is designed to exchange network reachability information between autonomous systems on the Internet.

Chapter 2

Background

2.1 Bitcoin

Bitcoin is a fully decentralized payment system. Every Bitcoin transaction is recorded in a public ledger called the *blockchain*. The blockchain consists of a sequence of validated *blocks*, which are groupings of transactions. Each block is marked with a timestamp and a link to its predecessor, thus the transaction history is built incrementally.

The shared state of Bitcoin is the unspent transaction output (UTXO) set. UTXOs consist of two parts: an amount of bitcoins, and a locking script that specifies conditions that must be met by the spending transaction. A *transaction* consists of one or more inputs and creates one or more outputs, which are confirmed by the whole network and available for the new owner to spend in a future transaction. Each input is the output of a previous transaction on the blockchain. No transaction input must have been spent in any other previous transaction on the blockchain, otherwise the transaction is not valid. Bitcoin uses ECDSA public key cryptography to establish proof of ownership; transactions are signed with the sender's private key. Transactions are broadcast over the peer-to-peer network and are included in new blocks.

Bitcoin uses a proof-of-work system based on computation. On average every 10 minutes a new block is created, i.e., a valid proof-of-work is found by a network node. The process of finding a valid proof-of-work is called *mining*. Miners are incentivized by receiving two types of rewards for creating valid blocks: newly created coins and transaction fees collected from the transactions that are included in the block. The amount of generated coins per block is halved every 4 years. This exponential decrease limits the total amount of bitcoins to 21 million, which will have been issued around the year 2140.

Because of the decentralized structure, blocks might arrive at different nodes at different times. Inconsistencies are possible in the form of competing chains (forks), since each full node has its local version of the blockchain. Majority consensus in Bitcoin is resolved by selecting the chain with the most proof of

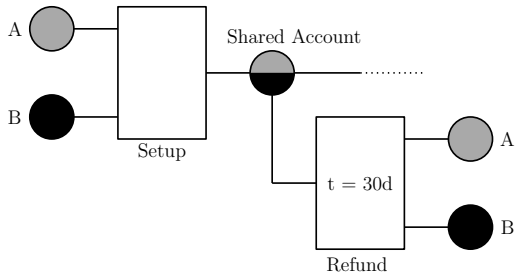


Figure 2.1: Setup of a shared account with a setup transaction and a refund transaction that becomes valid after 30 days.

work, i.e., the longest chain. This way inconsistencies are eventually resolved as more blocks are added to one of the forks.

2.2 Contracts, Shared Accounts and Micropayment Channels

Every Bitcoin transaction can have a locktime associated with it. Transactions will not be confirmed until the locktime. This makes a transaction replaceable until the agreed-upon time. The Bitcoin multi-signature feature allows the creation of m-of-n *multi-sig* outputs, requiring m signatures from a total of n public keys. A *2*-of-*2* multi-sig output requires both parties to agree on spending the output. Shared accounts are a form of contracts based on timelocks and multi-sig outputs. In order to create a shared account, two transactions are required: a setup transaction and a refund transaction. The *setup transaction* specifies as inputs the funds that both parties want to deposit on the shared account. The output is a *2*-of-*2* multi-sig output, the shared account. Before the parties sign this transaction, they create the *refund transaction*: a transaction that refunds multi-sig output to their original owners. This transaction is encumbered with a locktime in the near future, e.g., 30 days. Once the refund transaction has been signed by both parties, they are guaranteed to get a refund in case one becomes unresponsive. Finally, the setup transaction is signed and committed to the blockchain. This procedure is trustless and secure. In the worst case one's funds are locked up until the refund transaction becomes valid. Figure 2.1 depicts a shared account between two parties A and B.

Unidirectional micropayment channels enable a sender A to make small payments to a receiver B. Micropayment channels have two advantages over regular Bitcoin transactions on the blockchain: firstly, few commits to the blockchain are needed and secondly, the transfers are confirmed instantly. The setup consists of creating a shared account. In order to pay B, A creates a new transaction spending the funds of the shared account such that the new balance factors in

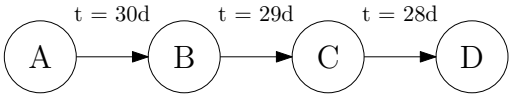


Figure 2.2: Multi-hop payment from A to D through a chain of hashed timelock contracts. Each node has a minimum time frame of 1 day to claim the previous output.

the payment. The transaction is partially signed by A. B can either claim the money right away by signing the transaction and committing it to the blockchain or she can wait in case she expects more payments from A. Every time a new transfer is made, the updated transaction takes the old balance and the payment into account. Unidirectionality follows from the fact that if B were to pay A, she could sign and commit a previous transaction received from A. If this transaction is included in a valid block before A's newer version of the transaction, the latest transaction from B to A is reverted.

Bidirectional micropayment channels such as Duplex Micropayment Channels [1], or as micropayment channels in the Lightning Network [5], enable two parties to make transactions in both directions. As with unidirectional micropayment channels, the amount a party in a bidirectional micropayment channel can send is limited by a capacity, i.e., one's funds on the shared account. For example, if A pays B, A's capacity on the micropayment channel decreases and B's increases. The initial funds for a bidirectional payment channel can be kept fairly small, if the two parties are expected to make small, frequent payments among each other. In order to prevent exhausted capacities, both parties should transfer roughly the same amounts over a period of time.

2.3 Hashed Timelock Contracts

In order to forward funds over a series of bidirectional micropayment channels, end-to-end security is required. The initial sender wants the guarantee that only the final recipient can claim funds. Hashed timelock contracts (HTLCs) satisfy this requirement. *Hashed timelock contracts* are contracts where the recipient can unlock an output with the knowledge of a secret S. The recipient B requests a payment from sender A by providing the hash $h(S)$ of secret S. A *settlement transaction* is the payment from A to B if B can provide the secret. A refund transaction is set up, encumbered with a timelock in the near future. Should the counterparty not be able to provide the secret S within a given timeframe, the refund transaction becomes valid. In the other case, the counterparties can agree on making an update transaction on their shared account. A third transaction, the *forfeiture transaction*, gives B the opportunity to free the funds of the HTLC output back to A even if the secret is eventually revealed.

End-to-end secure multi-hop payments can be made by chaining HTLCs together. Each HTLC along the chain is encumbered with the same hash *h(S)*. Figure 2.2 depicts a multi-hop payment from sender *A* to recipient *D*. Once a path from *A* to *D* is set up, *D* can claim the output by disclosing the secret *S* to its predecessor *C*. With knowledge of *S*, *C* can claim the HTLC output from its own predecessor *B*. Finally, *B* can claim the HTLC output from *A*, making the multi-hop payment complete. In order to guarantee security, the timelocks of the refund transactions of successive HTLCs along the path must be strictly decreasing. The difference between two successive timelocks must be big enough in order to give a node time to claim the previous HTLC output, i.e., the transaction needs to be confirmed.

Trustless, end-to-end secure, multi-hop payments can be made by using HTLCs on top of bidirectional micropayment channels.

Chapter 3

Protocol Design

The goal is to enable clients to make payments to each other that are routed over a network of payment service providers. PSPs operate in a similar way to autonomous systems on the Internet, and there may be PSPs whose sole business is to grant interconnection to other PSPs. A node in our network denotes a whole PSP. Whether or not two PSPs connect is a business decision and we assume it is negotiated directly. We concentrate on the routing part on an existing network.

To address sender and recipient we use payment protocol addresses (PPAs). PPAs differ from Bitcoin addresses and are rather similar to Internet Protocol addresses, e.g., IPv6 addresses. The key advantage is the ability to aggregate individual addresses to prefix sets, which allows routing based on network prefixes. One solution for the distribution of PPAs to the PSPs is the appointment of a central authority, similar to the Regional Internet Registries for the Internet. Another possibility is letting each PSP announce its network prefixes with the aid of cryptographic methods, e.g., randomly positioning in the address space by generating a matching public key.

It is in the PSPs' interest to incorporate fees. For example, in the Lightning Network fees are paid directly between counterparties within a channel. They pay for the time-value of money for consuming the channel for a determined maximum period of time, and for the counterparty risk of non-communication. Fees are simply implemented by transferring more money than necessary. Then, each PSPs along the routing path can claim a fraction of the fee amount specified by the sender.

Network nodes make routing decisions based on their routing tables. A routing table entry consists of a network prefix and a set of next hops with additional information to each hop, e.g., expected number of hops to the destination. A routing decision for a payment consists of one or more next hops, each with an associated amount. Payments can be routed over several next hops concurrently by splitting the amount, e.g., due to insufficient capacities. Heuristics and current measurements help a node in making routing decisions. For example, in figure 3.1 node A wants to make a payment of 1 BTC to node E. B's routing

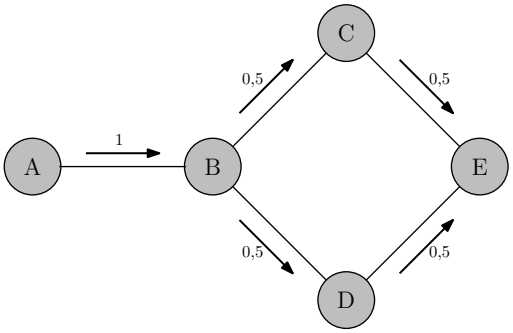


Figure 3.1: A payment is split along the path to the recipient.

table returns C and D as possible next hops for reaching E. B has not enough capacity to C nor to D in order to forward 1 BTC to one of them. Therefore, B decides to split the payment, sending 0.5 BTC to C and 0.5 BTC to D. It is in E's interest to wait until the full amount of 1 BTC is guaranteed. Once E can claim 0.5 BTC from C and 0.5 BTC from D, it discloses the secret to C and D.

The protocol is divided into two parts. The first is routing information exchange, enabling nodes to set up their routing tables. The second part is payment routing; nodes should be able to make routing requests to other nodes, decline routing requests and claim funds by settlement.

3.1 Routing Information Exchange

The *RoutingInfo* message (see figure 3.2a) allows nodes to announce reachable prefixes to their neighbors. *RoutingInfo* messages are broadcast periodically and for each known network prefix a set of routing paths is included. Initially, a routing table consists only of a PSP's direct neighbors.

On the receiver side, a node performs two checks on each path. First, paths whose length exceeds a threshold are discarded. The reason for this lies in the fact that we work with *time to live (TTL)* values in payment routing. In a second check, a node declines paths that already contain the node itself, such that no loops arise in the stored paths. New paths are added to the routing tables and included in future *RoutingInfo* messages. This incrementally builds a complete list of reachable subnetworks and corresponding paths.

Network nodes may disconnect. One possibility to remove paths including a disconnected node is to assign a timeout value to each path. The timeout of a path is renewed every time it is included in an incoming *RoutingInfo* message. Paths whose timeout has reached zero are eventually removed from the routing table. Another possibility is to ping nodes that are suspected to be offline. If the pinged node does not respond within a given time, paths including this node

```
message RoutingInfo {
    message Entry {
        string prefix;
        string path;
    }
    repeated Entry entry;
}
```

(a) *RoutingInfo* message

```
message Propose {
    string sender
    string recipient;
    string hash;
    int32 htlc_id;
    int32 amount;
    int32 fee;
    int32 TTL;
    string payload;
}
```

(b) *Propose* message

```
message NACK {
    string sender;
    string recipient;
    string hash;
    int32 htlc_id;
    int32 amount;
    string payload;
}
```

(c) *NACK* message

```
message ACK {
    string sender;
    string recipient;
    string hash;
    string secret;
    string payload;
}
```

(d) *ACK* message

Figure 3.2: The four protocol messages.

are removed from the routing table.

3.2 Payment Routing

Payments consist of a flow of chained HTLCs. A payment can be identified by the tuple *(Sender, Recipient, h(S))*, where *Sender* and *Recipient* denote the corresponding PPAs and *h(S)* denotes the hash of the secret provided by the recipient. A routing decision consists of one or more tuples *(next_hop, amount)*, where an *amount* is sent to a neighboring node *next_hop*.

The *Propose* message (see figure 3.2b) enables nodes to set up an HTLC. The first three fields *(Sender, Recipient, h(s))* identify the payment. Due to payment splits, multiple HTLCs may be set up between two nodes, all corresponding to the same payment. The *htlc_id* field associates every HTLC with a unique identifier, making HTLCs distinguishable. The *amount* denotes how much money is being routed. If a payment is split up, several *Propose* messages are sent to neighbors.

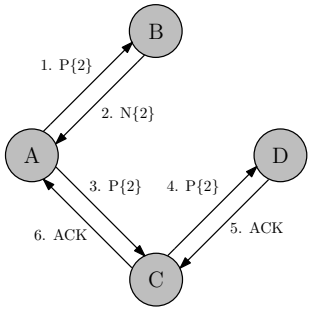


Figure 3.3: Node A initiates a payment of 2 bitcoins to D by sending a *Propose* message to B. Since B cannot reach D, it answers with a $NACK$ message. Then, the payment is routed over node C. Finally, D claims funds by sending an ACK message.

In the *fee* field, the inital sender can specify an amount that can be used for paying fees along the routing paths. It is in everybody's interest to claim only a fraction of this amount at each hop. Otherwise, if there is no amount left for fees, the payment may not reach the recipient. Although we check for loops in the routing information exchange, we cannot completely eliminate loops. A node does not know the prior path of an incoming *Propose* message. Therefore, there is no possibility to detect loops. Loops are undesirable, since they unnecessarily lock up funds until the corresponding HTLCs are cleared. For this reason we introduce *TTL*s in order to avoid infinite loops consuming all capacities. The TTL value is decreased at each hop, limiting the maximum HTLC chain length. Additional data can be added to the *payload* field, e.g., the partially signed Bitcoin transactions coordinating the HTLC.

In the case where a node cannot fully route a payment, e.g., due to exhausted capacities, it needs a way to cancel the previously set up HTLC by settling the forfeiture transaction, such that its predecessor can continue routing the amount. The $NACK$ message (see figure 3.2c) lets two nodes cancel a previously set up HTLC. Again, the first three fields identify the payment and the *htlc_id* field identifies a HTLC of a payment. The *amount* specifies how much money is forfeited. If there is a remaining amount, the recipient of the $NACK$ message responds with a *Propose* message, referring to the same *htlc_id*; a new HTLC with the remaining amount is set up. In order to guarantee security, the timelock of the new forfeiture transaction must be smaller than the timelock of the old settlement transaction. As with all HTLCs, the timelock of the new settlement transaction must be smaller than the locktime of the new forfeiture transaction. Several $NACK$ messages may be sent successively, each forfeiting a larger part of the total HTLC value. Every time the TTL value of a *Propose* message is zero, a $NACK$ message is sent back.

The payment recipient waits until the amounts of incoming *Propose* messages

sum up to the amount she is expecting, guaranteeing complete payment. Funds can be claimed by disclosure of S. The ACK message (see figure 3.2d) enables a node to disclose the secret to its predecessors. Two counterparties can then agree on making an update transaction on their payment channel. The update transaction can be included in the *payload* field. All HTLCs belonging to the same payment can be unlocked with the same secret.

Figure 3.3 shows a payment routed over a network with the aid of the protocol messages.

Chapter 4

Evaluation

4.1 Implementation

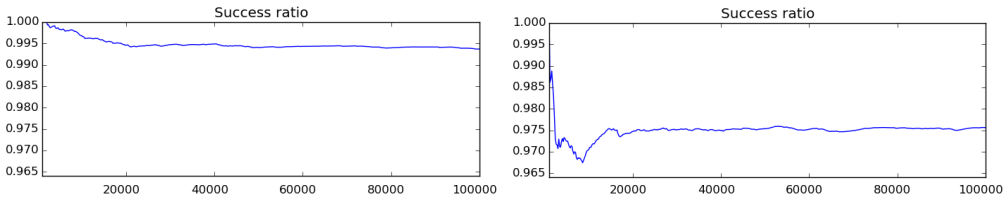
We implemented the protocol for both a real-world scenario and discrete event simulations. In the real-world scenario, nodes establish connection to each other through network sockets. Protocol messages are serialized and sent over the network. Larger networks were implemented in a simulation. We simulated the network layer, such that sending a message did not require sockets nor serialization. Small propagation delays were taken into account. This is important, so that HTLCs are active for some time and lock part of the capacities, otherwise HTLCs would be cleared immediately.

In both cases the dynamic parameters were:

- *Number of nodes*: the number of PSP nodes in the network.
- *Number of connections*: the minimum number of connections each node has. Nodes are randomly connected, taking the minimum number into account.
- *Channel capacity*: the capacity a node has on a micropayment channel. The amount is chosen uniformely at random from a specified range.
- *Time between two payments*: the expected average time between a node's two successive payments. After each payment, the time is chosen uniformely at random from a specified range.
- *Payment amount*: the expected amount per payment. The amount is chosen randomly, following the power law. It is scaled, such that the expected value of all amounts is equal to the desired amount per payment.

We let the nodes exchange *RoutingInfo* messages in order to build their routing tables. After some time, nodes would start making payments to each other.

The metric used for routing decisions was the number of hops to a destination, i.e., the length of the shortest path to the destination in the routing table. If



(a) *payment amount/capacity ratio*: 0.05 (b) *payment amount/capacity ratio*: 0.2

Figure 4.1: Simulations with different *payment amount/capacity ratios* on the same network topology. 64 nodes with at least 6 connections. On average, a node made 2000 payments during the simulation.

there were multiple equivalent candidates, the next hop was selected randomly. In case there was a remaining amount, the procedure of selecting a next hop was repeated. If a node would receive a payment expected to be routed to an unknown destination, i.e., a destination not included in its routing table, it responded with a *NACK* message. If a node received a *NACK* message from a neighbor, it excluded this neighbor from future *Propose* messages regarding this payment. Therefore, if a node received *NACK* messages from all its neighbors, and it could not route the whole amount, it would send back a *NACK* message itself. Apart from that, amounts from *NACK* messages were routed the same as amounts from *Propose* messages.

4.2 Measurements

Evaluation was performed on simulations with 64, 128 and 256 network nodes. A payment was considered successful when the full payment amount could be routed to the recipient. On the other hand, a payment was considered unsuccessful when the initial sender had to abort due to exhausted capacities, i.e., it was not possible to route the full payment to the recipient.

With fewer connections, local shortages are more likely to occur; more *NACK* messages are sent and payments take longer to be completed, which means that funds are locked up longer. More connections, on the other hand, generally require more initial funds for setting up the micropayment channels.

Our assumption that a higher ratio between average amount per payment and average capacity between nodes leads to a lower success ratio was supported by the simulation results. Figure 4.1 shows a comparison between different ratios on the same setting, i.e., same topology and same capacities. A higher *payment amount/capacity ratio* generally leads to more overall *NACK* messages, more unsuccessful payments and a higher average completion time for successful payments. In most cases, these numbers converged to a stable value, but there were also scenarios where no stabilization could be observed during the simulation time

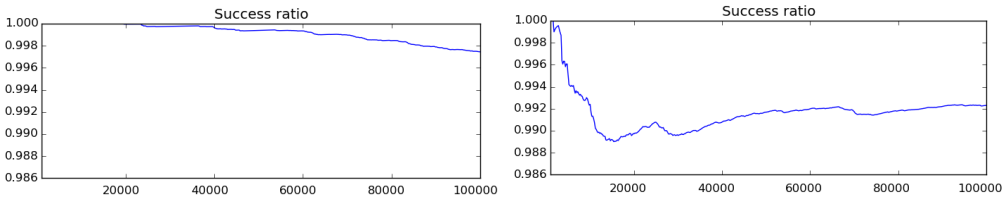


Figure 4.2: Two simulation with same parameters: *payment amount/capacity ratio* of 0.5, 64 nodes, at least 6 connections. On average, a node made 2000 payments during the simulation. On the left, the success rate slowly decreases with time while on the right, we see an early drop followed by a recovery phase.

frame. Simulations on the same setting with same *payment amount/capacity ratios* have been compared and showed that there can be notable differences in terms of success ratio. Figure 4.2 shows different success ratios for two simulations on the same setting. This behavior can be explained by local shortages, i.e., single nodes with exhausted capacities due to several big payments. Another point is that our routing strategy followed a greedy approach. Payments were split up where needed, capacities were fully exhausted if that meant fewer routing hops.

Simulations with a *payment amount/capacity ratio* around 0.05 had in most cases a high success rate above 0.99. For a PSP network of 64 nodes with 6 minimum connections and a capacity of 1 BTC for each micropayment channel, 778 BTC are needed. If we assume a *payment amount/capacity ratio* of 0.02, 1000 payments per second and a constant success rate of 0.98, payments amounting to 1 728 000 BTC could be made daily.

We predict an improvement by making smarter routing decisions, taking low capacities and other parameters into account. Dynamically increasing capacities could eliminate temporary shortages. Another point is connectedness: connections were selected at random; a more careful selection of the network topology is likely to improve results.

Chapter 5

Conclusion

A network of payment service providers, connected by bidirectional micropayment channels, provides a scalable environment to make end-to-end secure Bitcoin payments with instant confirmation.

The proposed protocol enables nodes to exchange routing information, similar to BGP for the Internet. Network nodes route payments by creating HTLCs using the protocol messages. Routing tables, heuristics and other measures can help the nodes in making routing decisions. Once a routing flow is set up, the funds can be claimed node by node until the initial sender is reached. We observed that the ratio between payment amount and channel capacity influences the ratio of successful payment transactions. Results can possibly be improved by a careful selection of network neighbors, dynamic capacity adjustment and more sophisticated routing strategies.

Bibliography

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

[2] Srdjan Capkun Ghassan O. Karame, Elli Androulaki. Two Bitcoins at the Price of One? Double-Spending Attacks on Fast Payments in Bitcoin. In *Proc. of Conference on Computer and Communication Security*, 2012.

[3] Mike Hearn. Bitcoin Contracts. https://en.bitcoin.it/wiki/Contract. [Online; accessed March 2015].

[4] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf. [Online; accessed March 2015].

[5] Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. https://lightning.network/lightning-network-paper.pdf. [Online; accessed March 2015].

[6] Yakov Rekhter and Tony Li. A Border Gateway Protocol 4 (BGP-4). https://tools.ietf.org/html/rfc4271. [Online; accessed April 2015].

[7] Yonatan Sompolinsky and Aviv Zohar. Accelerating Bitcoin's Transaction Processing.

[8] Lennart Elsen Roger Wattenhofer Samuel Welten Tobias Bamert, Christian Decker. Have a Snack, Pay with Bitcoins. In *IEEE International Conference on Peer-to-Peer Computing (P2P), Trento, Italy*, 2013.

[9] Peter Todd. Near-Zero Fee Transactions with Hub-and-Spoke Micropayments. http://sourceforge.net/p/bitcoin/mailman/message/33144746/. [Online; accessed August 2015].