Designing Efficient and Secure Algorithms for Payment Channel Networks

by

Sonbol Rahimpour

A thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Communications

Department of Electrical and Computer Engineering

University of Alberta

Abstract

Major blockchains such as Bitcoin and Ethereum suffer from scalability issues. For instance, Bitcoin can handle up to about 7 transactions per second, whereas custodian payment networks such as Visa can handle tens of thousands of transactions per second. One of the promising solutions to scale blockchains such as Bitcoin is Payment Channel Networks (PCNs). In this thesis, we propose new solutions to make the existing PCNs, such as the Bitcoin's Lightning network, more efficient and secure.

Chapter 3 studies watchtowers, which are entities that watch the blockchain on behalf of their offline clients to protect the clients' payment channels. In practice, there is no trust between clients and watchtowers, and it is challenging to incentivize watchtowers to well-behave (e.g., to refuse bribery). To tackle this problem, Chapter 3 proposes a novel reputation system, and uses the system to incentivize watchtowers to not only deliver their promised service but also reduce their service fees in competition with each other.

Chapter 4 proposes Spear, a new multi-path payment method that can support redundant payments. We show that this redundant payments can significantly improve the success rate of payment transfers in the Lightning network. The challenge in supporting redundant payments is that the recipient may overdraw funds from the redundant paths. In Chapter 4, we explain how this can be done without requiring high computation or major changes to the Lightning network.

Finally, Chapter 5 presents a powerful probing technique called Torrent

to discover balances of channels in the Lightning network. Unlike the existing techniques, Torrent uses multi-path payments instead of single-path payments. This—together with a novel max flow algorithm designed for the Lightning Network—allows a single probing node to push a large flow of payments through a target channel, making it possible to discover balances of even remote channels. Another major advantage of Torrent is that it can speed up balance discovery through parallel payment transfers and pre-computation of payment paths.

Preface

The results presented in Sections 3.1- 3.6 were published in IEEE International Conference on Blockchain and Cryptocurrency (ICBC) [1] in May 2021. The results of Chapter 4 were presented in the 3rd ACM Conference on Advances in Financial Technologies [2] in September 2021.

To my beloved
parents, sister and my lovely husband
without whom I would not have survived

Acknowledgements

First of all, I would like to express my sincere appreciation to my supervisor Dr. Majid Khabbazian who has helped me throughout my studies. Besides my supervisor, I am also thankful to my thesis committee members, Dr. Masoud Ardakani and Dr. Hai Jiang, for their comments and suggestions to improve my thesis. I would like to thank Dr. Chen Feng for serving as the external examiner for my final Ph.D. examination. I should also thank Dr. Marek Reformat for offering his time to read my thesis and agreeing to be an examiner for my final Ph.D. examination. I appreciate the advice of Dr. Ehab Elmallah who was my candidacy examiner. Also, I must thank all those whose actions, either directly or indirectly, have helped me to reach this point in my life.

I am incredibly grateful to my parents and sister for their love and spiritual support. Last but not least, I must express my deepest gratitude to my lovely husband for his love, encouragement, and support. Thank him for helping me to follow my dreams and believe in myself.

Contents

1 Introduction **1**

1.1 Incentivizing Watchtowers 2

1.2 Redundant multi-path payments 3

1.3 Balance Discovery in the Lightning Network 4

1.4 Thesis Contribution 6

1.4.1 Hashcashed Reputation 6

1.4.2 Spear: Fast Multi-Path Payment with Redundancy . . 7

1.4.3 Torrent: Balance Discovery in the Lightning Network Using Maximum Flow 7

1.5 Thesis Organization 8

2 Background **9**

2.1 Bitcoin and Blockchain 9

2.1.1 Bitcoin Transaction 9

2.1.2 Bitcoin Mining 10

2.2 Scalabality Problem and Solutions 10

2.2.1 Layer-one solutions: 11

2.2.2 Layer-two solutions: 11

2.3 Hashcash 12

2.4 Payment channels 12

2.5 The Lightning network 13

2.5.1 HTLC 14

2.6 Watchtowers 14

2.7 Maximum Flow 16

2.8 Related Work 17

2.8.1 Reputation systems 17
2.8.2 Proof of work 18
2.8.3 Payment channels and watchtower 18
2.8.4 Payment Routing 20
2.8.5 Balance Discovery attacks 21

3 Hashcashed Reputation **23**
3.1 System components 23
3.2 Interactions 26
3.3 Protocol 26
3.4 Adversarial model 29
3.5 Security Analysis 30
3.6 A reputation-based market of watchtowers 33
3.7 A reputation-based market of Timestamping 37
3.8 Conclusion 41

4 Spear: Fast Multi-Path Payment with Redundancy **42**
4.1 System Model 42
4.2 Power of Redundancy 43
4.3 Spear 47
4.3.1 Procedure 48
4.3.2 Security Guarantees 50
4.3.3 Implementation 51
4.4 Spear versus Boomerang 52
4.4.1 Latency 52
4.4.2 Implementation 57
4.4.3 Locktime 57
4.4.4 Computational overhead 58
4.4.5 Flexibility 59
4.4.6 Intermediate node's misbehaviour 59
4.4.7 Fees 60
4.5 Conclusion 61

5 Torrent: Balance Discovery in the Lightning Network Using Maximum Flow **62**

5.1 Torrent 62

5.2 Implementation 65

5.2.1 Network Model and Problem Definition 66

5.2.2 Linear Programming 66

5.2.3 Greedy Algorithm 67

5.3 Simulation Results 68

5.3.1 Linear Programming 69

5.3.2 Greedy Algorithm 70

5.4 Conclusion 71

6 Conclusions & Future Works **72**

Bibliography **75**

List of Tables

2.1 Existing balance discovery methods. 22

List of Figures

1.1 A simple probing technique for discovering Alice's Balance. . . 5

2.1 Simplified Bitcoin blockchain 10
2.2 The procedure of sending money from Alice to Bob through Charlie in the Lightning network. 15

3.1 An illustration of how a reputation is calculated. 24
3.2 A sample of watchtower contract. 35
3.3 An illustration of how a timestamp server creates a Merkle tree. 38
3.4 A sample of timestamp contract 39
3.5 The process of verifying a Proof-of-breach. 40

4.1 HTLC and H^2TLC in Bitcoin Script Pseudocode. 48
4.2 Making a payment in Spear. 50
4.3 Upgraded update-add-htlc message. Refer to Figure 4.4 for the `Onion_routing_packet` field. 52
4.4 The `onion_routing_packet` field. Refer to Figure 4.5 for the `hop_payload` field. 53
4.5 The `hop_payload` field. 54
4.6 The process for creating new commitment transactions 56
4.7 The process of creating an HTLC. 57
4.8 Transfer of a partial payment from S to P. The red dashed lines, and solid blue lines show locktime in Boomerang and Spear, respectively. 58

5.1 A small payment network. Carol has two channels with nodes n_1 and n_6, and is interested to discover balance of the channel between Alice and Bob. Balances of Alice and Bob on this channel are 200k and 150k Satoshis, respectively. 64
5.2 The percentage of discoverable channels by Torrents. 70
5.3 The percentage of discoverable channels by Torrents Type A and B when they use the greedy algorithm. 71
5.4 The optimal percentage of discoverable channels. 71

Chapter 1

Introduction

Permissionless blockchains can provide trust in a decentralized, trustless environment. This makes it possible for many applications such as financial applications to rely on a blockchain rather than trusted third parties such as banks. This shift in paradigm has brought a lot of interest and attention to blockchain. Blockchain applications such as Bitcoin are not, however, quite ready for mainstream use because they have scaling issues. Bitcoin, for example, can handle up to ten transactions per second, and needs, on average, at least ten minutes to add a transaction. Custodian payment systems such as Visa, on the other hand, can handle tens of thousands of transactions per second all with a short confirmation time.

Blockchain's moon race for a scalable solution has led to a rich body of literature, with solutions and proposals ranging from improving the consensus algorithms, to sharding [3], [4] and side-chains [5]. Among these solutions, "layer-two protocols" are among the most promising ones. These protocols allow users to conduct so-called *off-chain* transactions among themselves, without requiring to add these transactions to the blockchain. In this thesis, we focus on a class of layer-two protocols called payment channels.

A payment channel between two parties, say, Alice and Bob is essentially opened by publishing a contract that locks up a fund. This fund is called the channel capacity. For instance, if Alice and Bob each contribute 1 Bitcoin, then by publishing the channel contract, they can create a channel with capacity of 2 Bitcoins, with each party having a balance of one Bitcoin. After opening

the channel, Alice and Bob can transfer funds between themselves (without touching the blockchain) by simply agreeing on a new balance on the channel. These agreements are what is so-called off-chain transactions. When Alice and/or Bob decide to close the channel, they send their latest agreement to the contract. The contract, will then release the fund to each party according to the submitted agreement.

Payment channels guarantee the security of off-chain transactions using the locked-up fund. To maintain the security of the payment channel, however, parties need to be frequently online and watch the blockchain for possible frauds by the counter-parties; a party that goes offline risks losing payments as the counter-party can close the channel using an outdated channel state. A party who decides to go offline can employ a third-party, referred to as watchtower [6], to watch the blockchain and protect the channel on the party's behalf.

1.1 Incentivizing Watchtowers

Incentivizing watchtowers is a non-trivial task. One approach to incentivize watchtowers is to pay them at the beginning of the watching service we expect them to provide in the future. The challenge with this approach is to make sure that watchtowers will, in fact, fulfill the service for which they were paid. There are known solutions [7], [8] that can handle this using complex smart contracts. Such solutions are, however, not applicable to Bitcoin's simple script language. In addition, they generate extra load on the blockchain, which is what payment channels try to avoid in the first place. Another approach is to pay watchtowers a fee only when they detect a fraud [9], [10], [6]. A major issue with this approach is that watchtowers will not receive any fee if (thanks to them) no fraud occurs. In fact, this approach may incentivize watchtowers to encourage frauds. In Chapter 3, we tackle this problem by designing a novel reputation system and using it to incentivize watchtowers.

1.2 Redundant multi-path payments

Payment channels can be connected to each other to form a network. An example of such payment network is the Bitcoin's Lightning network. In the Lightning Network, Alice can use her channel with Bob to make a payment to another party, say Carol, who is already connected to the network. To perform this, Alice first selects a path of payment channels P from herself to Carol, and then transfers the payment to Carol through P. Because of privacy concerns, the balance of channels in the Lightning Network are considered private information. Since Alice is not aware of balances of channels on P (except the first channel on P which belongs to her), her payment may fail because a channel on the path may not have enough balance to forward the payment. If Alice's payment fails, she should retry the transfer by choosing a different path. This adds a non-negligible delay to the payment process, and increases the payment's time-to-complete.

To reduce the chance of payment failure (hence, reducing the time-to-complete), Alice may decide to split the payment into partial payments and transfer these partial payments through multiple paths. On the one hand, this helps as the smaller a payment, the more likely it is for the payment to go through. On the other hand, the whole payment will fail if any of the partial payments fails. In Chapter 4, we show that this multi-path payment approach does not necessary help unless we use redundant paths. The main challenge in adding redundancy in a multi-path payment method is to protect Alice from overdrawing.

In [11], Bagaria, Neu, and Tse introduced Boomerang, the first multi-path payment method with redundant payments, and showed that it can improve throughput and time-to-completion of payments. To address the overdrawing problem, Boomerang uses secret sharing to enable Alice to recover the payment when Bob overdraws.

In Chapter 4, we will take a simpler approach, called Spear, to add redundancy and protect Alice against overdrawing. While Boomerang's approach resembles coding techniques in data storage systems and communications, our

solution Spear resembles ARQ (Automatic Repeat Request), a method that uses acknowledgements to achieve reliable data transmission over an unreliable communication channel.

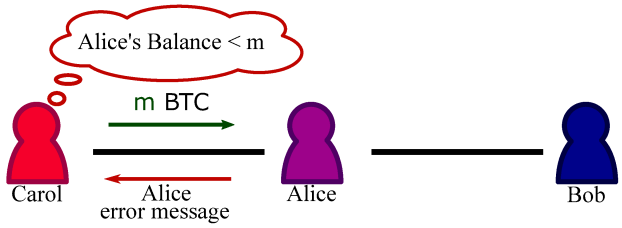
1.3 Balance Discovery in the Lightning Network

The Lightning Network offers higher privacy than the Bitcoin blockchain. To enhance privacy, the Lightning Network employs several mechanisms including *onion routing* and *balance concealment* [12]. The onion routing hides the sender and the recipient of a payment by encapsulating the payment information in layers of encryption. As a result, each node on the payment path can identify only its immediate predecessor and successor.

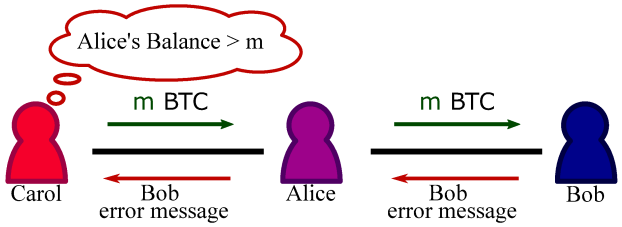
A successful payment will change the balance of all the channels on its path from the sender to the recipient of the payment. Therefore, if channel balances are made public, any network observer can extract payment transfer information (including the source, destination, and the value of a payment) by simply observing changes in the balance of channels. To prevent this, the Lightning Network requires nodes to conceal the balance of their channels from others.

Although balances are kept secret, several recent studies have shown that they can be discovered or at least estimated using simple probing techniques: suppose Carol is interested to know Alice's balance on a channel between Alice and Bob. A basic approach described in [13], is for Carol to first open a channel with Alice. If successful, as shown in Figure 1.1, Carol then sends a "fake payment" to Bob through Alice. Such a fake payment will trigger an error message. If Carol receives the error message from Alice, she concludes that Alice's balance is lower than the amount of payment because only Bob, the recipient of the payment, can determine that the payment is fake. Otherwise, if Carol receives the error message from Bob, she knows that the payment has successfully passed through the Alice-Bob channel, thus Alice's balance must be higher than the value of the payment. Carol can repeat sending fake

(a) Carol sends a fake payment (with value m Bitcoin) to Bob through Alice.



(b) Carol receives an error message from Alice.



(c) Carol receives an error message from Bob.

Figure 1.1: A simple probing technique for discovering Alice's Balance.

payments with different values (following, say, a binary search pattern) to Bob, improving her estimate on Alice's balance with transmission of each such fake payment.

The above balance discovery method does not work if neither Alice nor Bob accepts Carol's request for opening a channel. Furthermore, the method does not scale: if Carol wants to discover balances of many channels, she has to open many channels, too. This is costly as Carol has to lock funds and pay transaction fees for opening each channel.

Recent extensions [14]–[16] have shown that Carol does not need to open a channel with Alice to discover her balance. Instead, she can use any channel, and send fake payments using a path $\mathcal{P}$ that goes through the Alice-Bob channel. If she receives an error message from Bob or any other node after Bob on path $\mathcal{P}$, she knows that the payment has passed through the Alice-Bob channel, hence Alice's balance must be higher than the payment's value. If she receives an error message from Alice, on the other hand, she can conclude

that Alice's balance was not enough to let the payment go through. If Carol receives an error message from a node that is before Alice on path $\mathcal{P}$, however, she will not gain any information about Alice's balance. In other words, Carol cannot gain any information on Alice's balance if she cannot push enough flow of money to Alice through $\mathcal{P}$. Of course, Carol can try again by choosing a different path. However, there is no guarantee that the new path can carry enough flow to Alice. In fact, there may be no single path that can carry enough flow from Carol to Alice.

Existing extensions use a sequential probing process in which fake payments are sent sequentially. This slows down the balance discovery process. This is not desired because Carol may need to know balances as soon as possible since the Lightning Network is dynamic and balances can change quickly.

To address the above shortcomings, we present *Torrent* in Chapter 5. Unlike the existing methods, Torrent can concurrently send several payments through multiple paths. These payments are temporarily held at a receiving node to prevents payment cancellations during the balance discovery process. This technique allows Carol to quickly push a large flow of money through Alice using concurrent payments. Using Torrent, Carol can discover Alice's balance if the maximum flow that she can bring to Alice is larger than Alice's balance. To achieve the maximum flow (or get close to it), we devise new algorithms that work within the unique limitations of the Lightning Network and our specific channel discovery problem. We show that, using these algorithms, Carol can quickly discover balances of many remote channels with opening at most two channels.

1.4 Thesis Contribution

1.4.1 Hashcashed Reputation

In Chapter 3, we propose a novel reputation system to stimulate well-behaviour, and competition in online markets. Our reputation system is suited for markets where a publicly-verifiable "proof-of-misbehaviour" can be generated when one party misbehaves. Such markets include those that provide blockchain

services, such as monitoring services by watchtowers and blockchain-based timestamping.

1.4.2 Spear: Fast Multi-Path Payment with Redundancy

In Chapter 4 we evaluate and compare the success probabilities of multi-path payment with and without redundancy, and demonstrate the superiority of the former. We introduce Spear, a simple multi-path payment method with redundancy. We prove Spear's various security guarantees, and explain how to implement Spear. Moreover, we compare Spear with its counterpart, Boomerang, against several measures including delay (i.e., time-to-complete), computational complexity, ease of implementation, maximum required lock-time, resilience against intermediate node's misbehaviour, and flexibility. Our comparison results show that Spear improves Boomerang in all these measures, hence is a better choice in applications where there is an out-of-band channel between the payer and payee.

1.4.3 Torrent: Balance Discovery in the Lightning Network Using Maximum Flow

In Chapter 5, we present a more powerful probing technique called Torrent. Unlike the existing techniques, Torrent uses multi-path payments instead of single-path payments. This—together with a novel max flow algorithm designed for the Lightning Network—allows a single probing node to push a large flow of payments through a target channel, making it possible to discover balances of even remote channels. Another major advantage of Torrent is that it can speed-up balance discovery through parallel payment transfers, and pre-computation of payment paths. In addition, Torrent can work efficiently in the absence of routing messages that indicate insufficient balances.

1.5 Thesis Organization

The remaining chapters of this thesis are organized as follows. Chapter 2 provides the necessary background. Chapter 3 deals with proposing a new reputation system and using it to create a watchtower market. Chapter 4 evaluates and compares our fast multi-path payment with redundancy method with previous methods with or without redundancy. Chapter 5 uses a novel max flow algorithm to discover balances of channels. Finally, Chapter 6 concludes this thesis and presents the possible extensions to our works.

Chapter 2

Background

2.1 Bitcoin and Blockchain

Bitcoin is the first decentralized digital currency proposed by Satoshi Nakamoto in 2008. Bitcoin allows users to transfer currency in its network without a trusted third party. In the Bitcoin network, nodes (miners) verify and record transactions using a distributed ledger technology called blockchain. A blockchain is created by blocks that are linked together by hash values (Figure 2.1). Each block includes some transactions' data and a hash value of the previous block. Other blockchain applications are secure sharing of medical data [17], NFT marketplaces [18], real-time IoT operating systems [19], voting mechanisms [20], and real estate processing platforms [21].

2.1.1 Bitcoin Transaction

A bitcoin transaction contains inputs and outputs to determine where the currency comes from and goes, respectively. The sender (or receiver) in a transaction is defined by an address. In Bitcoin, users create their own addresses by generating private and public keys. The transaction's outputs include the amount of bitcoin and conditions for spending. After users create a transaction, they broadcast it in the Bitcoin network. Upon receiving a transaction, nodes verify it. If the transaction is valid, they keep a copy of the transaction for themselves and broadcast it to other nodes.

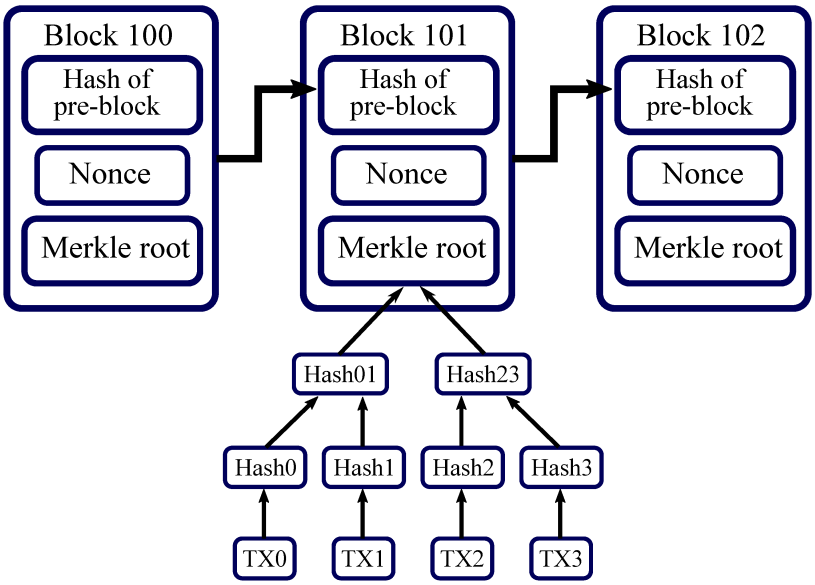


Figure 2.1: Simplified Bitcoin blockchain

2.1.2 Bitcoin Mining

To create a block, a miner creates a Merkle tree [22] by using a set of valid transactions as leaves (Figure 2.1). The root of this Merkle tree is then placed in a header. The miner also places other information such as the hash of the previous block in the header. After this, the miner starts the proof-of-work process by tyring different values of nonces until the hash of the header is smaller than the protocol's difficulty target. The first miner who reaches the difficulty miner will broadcast its created block to other miners, who will verify the whole block (transactions and proof-of-work) and will add the newly minded block to their chain.

2.2 Scalability Problem and Solutions

Currently, because of the difficulty of the mining problem in Bitcoin, the interval time between generating two blocks is about ten minutes. In other words, each transaction needs ten minutes on average to be published. Furthermore, the size of blocks in Bitcoin should be at most 1MB, which limits the number of transactions that can be placed within one block. As a result, the Bitcoin network can process up to about 7 transactions per second since each

transaction is on average about 384 bytes. In comparison, custodian payment methods (such as Visa), can quickly confirm transactions and can handle tens of thousands of transactions per second. Therefore, Bitcoin needs to scale before it can get widely adopted as a payment system. The current solutions to scale Bitcoin (as well as other blockchains such as Ethereum) can be placed in two classes: Layer-one and Layer-two.

2.2.1 Layer-one solutions:

Layer-one solutions try to improve the blockchain's architect (such as the block structure, and consensus algorithm). One such solution is to increase the size of the block or decrease the interval between blocks, as it is used in Bitcoin Cash [23]. The problem for solutions with larger blocks is that the blocks need more time to propagate through the network, which causes occurring more forks in the blockchain. Therefore, the probability of a double-spend attack is increased [24].

Another approach to scaling blockchain is *sharding* [4], [3]. This idea has roots in database sharding, where data is divided into shards, and shards are placed on different servers. However, sharding increases the complexity of the system.

2.2.2 Layer-two solutions:

Layer-two solutions concentrate on off-chain methods that move the massive number of transactions or computational tasks off the blockchain. One of these methods is payment channel networks such as Bitcoin's Lightning network. In this approach, users can exchange one million transactions per second without submitting them to the blockchain. Another approach in Layer-two solutions is state-channel [25], [26], [8]. State-channels are the general form of payment channels. In state-channel solutions, users move their interactions (besides their transactions) into off-chain. Side-chains such as [27] are another solution that use a series of contracts on the main chain. We refer the reader to [28] for more details about these solutions.

In addition to the scalability issue of Bitcoin, Bitcoin has a high environmental cost, transaction fee, delay at processing transactions, and it is not refundable.

2.3 Hashcash

Hashcash is a type of proof-of-work that is based on cryptographic hash functions such as SHA-256. Let H be such a cryptographic hash function. For a given string s, we say h is an n-bit hashcash of s, if $H(s||h)$ in binary has n leading zeros, where $s||h$ indicates the concatenation of s and h. One can easily verify a hashcash h by computing $H(s||h)$, and then counting the number of leading zeros of the result. To find an n-bit hashcash, however, one requires to compute the hash function 2^n times, on average. Therefore, finding an $(n+1)$-bit hashcash requires, on average, twice as much work as finding an n-bit hashcash. The value of n is used as the measure of the amount of work performed.

2.4 Payment channels

Payment channels [29], [12] are a promising solution to the low throughput and high latency of Bitcoin. Payment channels achieve this by handling most transactions outside the Bitcoin blockchain. A payment channel can be viewed as a temporary joint account between two parties, say Alice and Bob. The channel is opened by a Bitcoin transaction, referred to as the opening transaction (*topen*). The opening transaction commits the parties UTXOs (Unspent Transaction Outputs) into a single 2-of-2 multisig output, controlled jointly by Alice and Bob. Once the channel is opened, Alice and Bob can exchange private transactions off the chain. These off-chain transactions, called commitment transactions (*ctx*), commit the output of *topen* into a set of outputs that divide the channel fund between Alice and Bob.

For example, suppose Alice and Bob each deposit two Bitcoins to open a payment channel. The first commitment transaction, ctx_1, divides the total channel fund of four Bitcoins evenly between Alice and Bob. This commitment

transaction ensures that each party can get their money back in case the counter-party disappears after the channel is opened. Now, suppose that Bob wishes to purchase a good worth of one Bitcoin from Alice. To make the payment, Bob provides Alice with a new commitment transaction, ctx_2, which updates the channel balance by giving Alice three Bitcoins and giving Bob one Bitcoin.

In addition to this, Bob must revoke ctx_1, as this transaction now reflects an outdated balance. To this end, Bob will give Alice a so called justice transaction jtx_1. The justice transaction is used by Alice to penalize Bob if he publishes ctx_1 in the blockchain. To enable this punishment mechanism, commitment transaction is designed such that once published by one party, they give the counter-party a dispute period during which the counter-party can send a justice transaction if there is any. In our example, if Bob publishes ctx_1, Alice can dispute it using jtx_1 and collect the whole channel fund.

2.5 The Lightning network

Two parties may not have a payment channel between themselves, but they may be connected through multiple payment channels. For example, Alice may not have a payment channel with Bob, but she may have a payment channel with Charlie, who has a payment channel with Bob. In this case, the Lightning network enables Alice to transfer money to Bob through Charlie.

The challenge to transfer money from Alice to Bob through Charlie is that the transfer from Alice to Charlie and the one from Charlie to Bob are independent. Consequently, if one of these two transfers goes through, there is no guarantee that the other one will go through. The Lightning network handles this issue by binding the two transfers using a Hashed TimeLock Contract (HTLC). Using HTLC, the two transfers on the way from Alice to Bob are conditioned on Bob releasing a secrete preimage. This essentially ensures that Bob's secret preimage and Alice's money are atomically exchanged.

2.5.1 HTLC

A Hashed Timelock Contract is a type of smart contract that enables the implementation of time-bound transactions between two nodes in the Lightning network. This contract pays money to a payee in a condition that the payee must satisfy before a particular time (timelock).

Consider the previous example in which Alice wants to send money to Bob through Charlie. Bob first generates a random number R (called preimage) and computes its hash image $H(R)$. Then, he gives this hash image to Alice. Bob can use this hash image as part of an invoice. In response, Alice creates an HTLC with Charlie. For creating the HTLC, Alice sends a message to Charlie. This message includes the amount of money, $H(R)$, timelock, and some information about the following users in the path. Based on this HTLC, Charlie receives her money if she gives Alice R within a specific timeframe.

After creating HTLC between Alice and Charlie, Charlie creates an HTLC with Bob. The hash image in HTLC between Charlie and Bob is also $H(R)$. If Bob reveals R to Charlie by the timelock, he receives money from Charlie, in which case Charlie can use R to claim her money from Alice[1] (we represent this example in Figure 2.2). The preimage can be acted as a receipt or proof of payment.

2.6 Watchtowers

The punishment mechanism explained earlier requires each party to stay online and monitor the blockchain for possible cheating by the counter-party. Alternatively, a party may delegate the task of monitoring the blockchain to a third party called watchtower. In practice, this is accomplished by giving the watchtower the first 16 bytes of every ctx's transaction ID (ctx_{txid}), as well as every justice transaction encrypted using the second 16 bytes of the corresponding ctx_{txid}. If the watchtower finds a transaction on the blockchain with an ID whose 16-byte prefix matches a prefix that it has stored, it will decrypt

[1]The timelock in this HTLC is less than the timelock in the HTLC between Alice and Charlie. By this method, when Charlie receives R from Bob, she has enough time to claim her money from Alice.

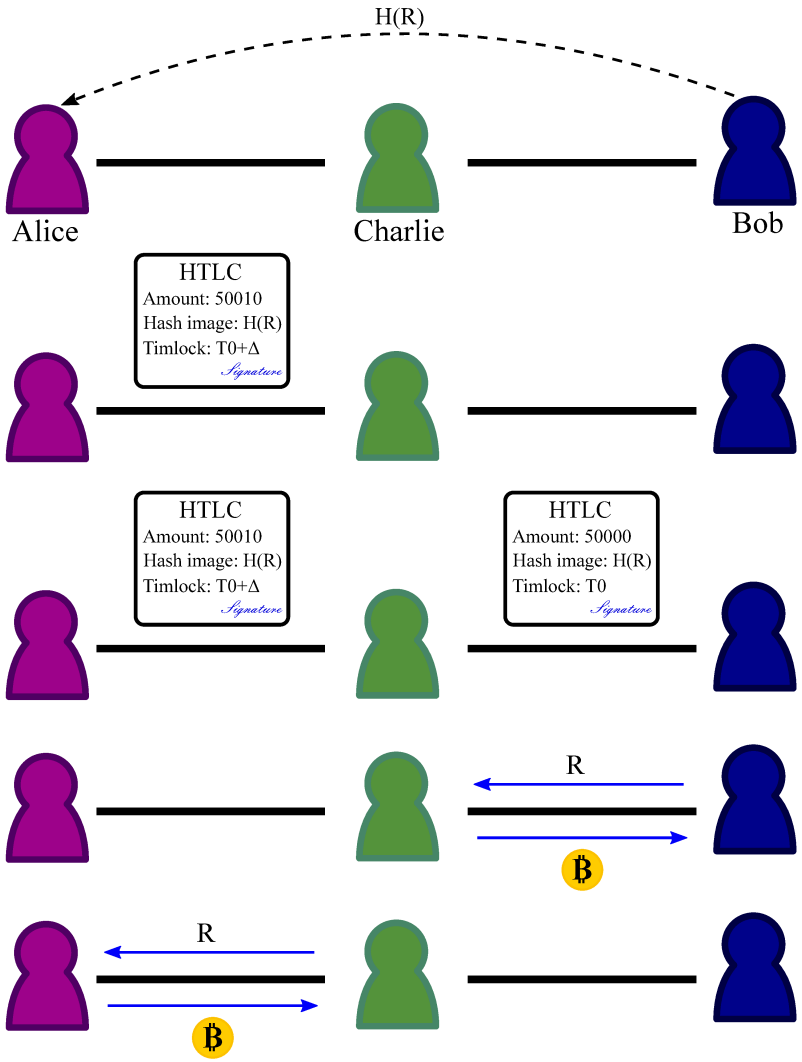


Figure 2.2: The procedure of sending money from Alice to Bob through Charlie in the Lightning network.

the corresponding jtx transaction using the second 16 bytes of the transaction ID, and then broadcasts jtx to the network to penalize the cheating party. Note that a watchtower cannot identify a channel in this design unless one of the two channel's owners cheats.

There is a special case where one of the two parties does not need to watch the blockchain, hence does not require a watchtower when the party goes offline. This case is when Alice opens a payment channel with Bob and uses this channel only to pay Bob. In other words, Alice does not receive/accept any payment from Bob on the channel. We call such a channel a *directional payment channel* from Alice to Bob. For a directional payment channel from Alice to Bob, Alice does not need to watch the blockchain for old commitment transactions. It is because Bob does not have any incentive to cheat, as old transactions give Bob less money than the latest commitment transaction. Note that unlike Alice, Bob needs to watch the blockchain as Alice has an incentive to cheat by claiming an old commitment transaction.

2.7 Maximum Flow

The maximum flow problem is about finding the maximum amount of feasible flow that a node (source) can send to another node (sink) in a flow network. A flow network is a directed graph, where each edge has a capacity that restricts the maximum flow that can pass through the edge. The amount of flow that enters a node (other than the source and the sink) must be equal to the amount of flow that leaves the node.

The maximum flow problem was first formulated in 1954 by Harris and Ross for Soviet railway traffic flow [30]. Soon after, Ford and Fulkerson proposed the first algorithm to solve the problem [31]. Since then many improved solutions with lower computational complexities have been proposed. These solutions are used today in various applications including mining industry [32], optimizing spatiotemporal data scheduling methods [33], optimizing complex transactions in a traditional bank accounts, Bitcoin wallet accounts and Bitcoin exchanges [34] and controlling power transmission [35].

Despite the existing solutions, solving the maximum flow problem in the Lightning Network is challenging because balances are not known. In the absence of balance information, we can access an oracle[2]. The oracle can tell us whether a given path can carry a given flow, and if so, can be asked to make the payment and update balances of the channels on the given path.

2.8 Related Work

2.8.1 Reputation systems

Different reputation systems use different methods to calculate reputation. We classify these methods into fact-based, and review-based methods. Fact-based methods calculate the reputation of a party by solely taking the activities of the party as input. Review-based methods, however, calculate the reputation of a party using reviews the party receives from other parties.

Fact-based methods. These methods evaluate the performance of a party using a predefined function that takes the party's activities as input [36]–[42]. Therefore, in fact-based methods, the reputation of a party is only a function of its activities. For example, in [36] the reputation of a party is increased or decreased based on the value of transactions on which the party was honest or dishonest, respectively. Another example is CrowdBC [37], a reputation system with two types of parties: requester and worker. A requester is a party that offers a task, while a worker is a party that performs tasks to improve its reputation. In CrowdBC, a requester and a worker negotiate and generate a smart contract. This contract has an evaluation-function that evaluates the performance of the worker. CrowdBC uses the output of this evaluation-function as well as the average reputation of all other workers to calculate the reputation of the worker.

Review-based methods. In this class, reputation of a party depends on the reviews and scores it receives from other parties [43]–[45]. For example, in online shopping stores such as eBay and Amazon, buyers can review and rate sellers and items. These rates together represent the reputation of sellers

[2]The oracle is the Lightning Network itself.

and items. Another example is Kudos [45], which is an educational reputation currency in a blockchain that records intellectual efforts, and related reputation rewards. Academic people and institutions that award certificates or verify innovations are parties of Kudos. When a person completes a certificate, their institution sends them some amount of Kudos based on their review. In this system, the amount of Kudos a party has represents the reputation of the party.

2.8.2 Proof of work

Dwork and Naor [46] introduced the concept of proof-of-work in 1992. Motivated to combat junk emails, they proposed to require users to compute a moderately hard function of their messages and some additional information in order to send their messages. They called these functions *pricing functions*, and introduced several of them in their work. In 1997, Back [47] proposed Hashcash proof-of-work system to deter email spams, and denial-of-service attacks. The Hashcah system is used today as part of consensus protocols in many blockchains, including Bitcoin [48].

In addition to combating denial-of-service attacks (e.g.,[49]), proof-of-work has been used to mitigate Sybil attacks [47], [50], [51], [46], [52], protect peer-to-peer resource sharing [53], and reward well-behaving users [54]. For instance, in [54], the authors proposed a micropayment method to reward Tor relay operators. In the Tor network, selfish clients may utilize the shared bandwidth of Tor relays without contributing any resources to the system in return. To mitigate such selfishness, Tor clients must submit proof-of-work shares, which Tor relays can resubmit to a cryptocurrency mining pool instead of paying cash directly. By analyzing the cryptocurrencies market prices, the authors showed that their method can compensate for a significant part of the Tor relay operator's expenses.

2.8.3 Payment channels and watchtower

Payment channels were first introduced by Satoshi Nakamoto [55]. These channels first emerged as unidirectional for one-way payments [56], then transi-

tioned into bidirectional channels to support two-way payments. The two common bidirectional payment channels are the Lightning Network [12] and the Raiden Network [26] which operate on Bitcoin and Ethereum blockchain [57], respectively. There are several implementations of the Lightning network, including C-lightning [58], Eclair [59], and LND [60].

To secure payment channels, users must frequently be online and watch the blockchain to protect their funds. It is because one party may publish an old commitment transaction while the other party is offline. Dryja [10] suggested that users who decide to go offline for an extended period of time delegate the task of watching the blockchain to third parties. Hertig [6] called these third parties *watchtowers*. Designing a secure, efficient, and decentralized watchtower protocol is a challenging task.

McCorry *et al.* proposed Pisa [7], a protocol that employs third parties called custodian to protect Sprites channels [8]. In Pisa, users pay their custodians every time they make a transaction on the channel. On the other hand, custodians lock a collateral fund, which they lose if they misbehave.

In contrast to Pisa, which requires complex smart contracts, our protocol does not rely on any smart contract. Avarikioti *et al.* proposed the DCWC protocol [9], in which full nodes can act as a watchtower for multiple channels. Unlike Pisa and our protocol, in DCWC, a watchtower gets paid only when it catches a fraud. In another research work, Avarikioti *et al.* presented BRICK [61], a protocol that detects and prevents fraud before it appears on the blockchain. To this end, BRICK employs a committee of third parties called Wardens. Wardens confirm the validity of each state channel and make sure that only the correct state is published in the blockchain when a dispute occurs.

The authors in [62] proposed Outpost, a lightweight structure for watchtower that encodes justice transactions within commitment transactions rather than storing them in the watchtower. This construction saves an order of magnitude in storage over existing watchtower designs. Finally, Avarikioti *et al.* extended the Lightning network and introduced Cerberus Channels [63]. They motivate watchtowers to work honestly by rewarding them for any update on

Cerberus Channels and forcing them to lock a collateral and pay a penalty when a fraud occurs. Increasing locking collateral, however, decreases the motivation of watchtowers to accept the work. In our model, watchtowers do not have to lock money. In case of a fraud, however, they are punished by losing their reputation.

2.8.4 Payment Routing

There are three main classes of payment routing methods [28]: single-path payment [12], multi-path payment [64]–[66], and packet-switch routing [67]–[69]. In the single-path routing, Alice sends the whole payment through a single path. In multi-path payment, Alice can split her payment into partial payments and transfer them through different paths. Lastly, in the packet-switch routing, Alice splits the payment into unit of payments and sends them individually.

The multi-Path Payment (MPP) method [64] is the simplest type of multi-path payment proposed for the Lightning Network. In MPP, Alice splits the payment into partial payments and sends them through different paths. All these partial payments are conditioned on the disclosure of a single preimage known by Bob. When Bob receives all the partial payments, he can accept/settle all these payments by disclosing his preimage.

Another existing multi-path payment method for the Lightning Network is Atomic Multi-Path Payment (AMP) proposed in [70]. In AMP, Alice uses a "base preimage" for the payment, from which she derives payment preimages of partial payments. As in MPP, Alice splits the payment into partial payments, and transfers these payments through different paths. However, in AMP, Bob needs to receive all the partial payments in order to construct the preimages and claim the partial payments. Therefore, unlike MPP, in AMP Bob cannot claim any partial payment before he receives all the partial payments.

In both MPP and AMP, the whole payment fails/delays if any of the partial payments fails/delays. Packet-switch routing methods [66], [71]–[75] can alleviate this to some extent. For example, in Ethna [76], and Interdimensional Speedy Murmurs [72] protocol and its extension [66], in addition to Alice, in-

termediate nodes are able to split a payment. Furthermore, these protocols allow intermediate users to select the next user on the path. However, as in MPP and AMP, the whole payment fails/delays if any single partial payment fails/delays. In addition, packet-switch routing methods may require a high number HTLCs. This can put pressure on the network as each channel can support only a limited number of (unsettled) HTLCs.

One can consider a fourth class of payment methods: multi-path payments with redundancy. As far as the authors know, Boomerang [11] is the only existing method that falls within this class. In this work, we propose Spear, the second method within this class. As shown in [11] and this work, these methods have a great potential in improving the performance of multi-path payment.

2.8.5 Balance Discovery attacks

Table 2.1 compares the existing balance discovery methods. As indicated in the table, the methods proposed in [13], [77], [78] require opening a channel directly connected to the target channel. This may not be possible in practice as the owners of the target channel may not accept new channels. In addition, these methods are not scalable because they have to open a new channel for every target channel.

The remaining balance discovery methods [14]–[16] do not require direction connection to the target channel. However, they all use single-path payments, and sequentially probe the target channel. As mentioned earlier, single-path payments can generate a limited flow compared to multi-path payments, hence have limited capability to discover balances of remote channels. Moreover, these methods are slower than Torrent (the method we will explain in Chapter 5) as they do not transmits payments in parallel.

Method	Need a direct channel?	Relies on error messages?	Success rate
Balance discovery attack [13]	Yes	No	89.1%
Improved balance discovery attack [77]	Yes	No	98.37%
Generic attack [78]	Yes	Yes	No Data
Probing channel balances [14]	No.	No	65%
Probing attack [15]	No	No	No Data
Probing of Parallel Channels [79]	No	No	80%
NIBT [16]	No.	Yes	92%
Torrent	No.	No	87%–97%

Table 2.1: Existing balance discovery methods.

Chapter 3

Hashcashed Reputation

In this chapter, we propose our Hashcashed reputation system, and show how the system can be used to create a market that stimulate well-behaviour and competition. To showcase our reputation system, in this chapter, we create an open market of watchtowers, where watchtowers are motivated to not only deliver their promised service but also reduce their service fees in competition with each other.

3.1 System components

Market. A market is identified by a number called the *market ID*. The main purpose of this number is to distinguish different markets, hence it can be set by, for example, a centralized organization that oversees global market ID allocation. A market is composed of servers, which are entities that provide a defined service to clients for profit. The markets we consider are open, which means they allow any server to join and offer their service. Every server is identified by an ID, which is a public key. The market has no central authority to assign IDs to servers. Therefore, similar to Bitcoin users, servers select their IDs on their own. As a result, an entity can enter the market with many different IDs, as it can create many public keys.

Reputation. In our system, a server generates its own reputation and proves it using a hashcash. More specifically, given a server ID, a market ID, and a hashcash the reputation of a server is calculated as the number of

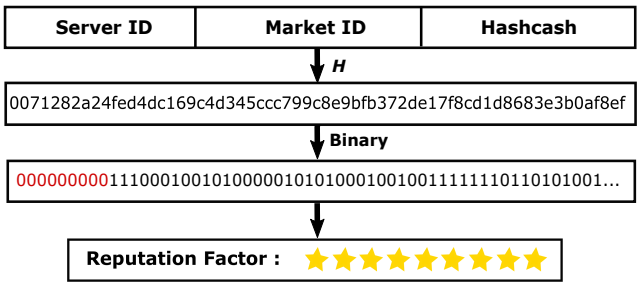


Figure 3.1: An illustration of how a reputation is calculated.

leading zeros of

$$H(serverID||marketID||hashcash)$$

in binary, where H is a cryptographic hash function (such as SHA-256). This is illustrated in Figure 3.1. Note that the hashcash is basically a nonce, similar to the nonce in the Bitcoin block headers. Also, note that reputation is tightly linked to a single pair of server ID and market ID. This prevents an entity from using a reputation for multiple IDs or over multiple markets.

A server can increase its reputation on its own by creating a better hashcash. This is in contrast to the existing reputation systems, where a server's reputation is increased when it receives good reviews from clients/customers. Finding a better hashcash, however, is not free. The server has to mine itself, or rent mining power (whichever the server finds more cost-effective). A server may make such an investment to, for example, get a better share of the market or merely maintain its share in a competitive (but profitable) market.

Definition 1 (Reputation cost) *The reputation cost, $cost(r)$, is an estimate of the minimum energy (electricity) cost to generate a server ID with reputation r. The cost of a server s with reputation r is denoted $cost(s)$, and is defined to be equal to $cost(r)$. We remark that reputation cost is time variant, because energy cost and hardware efficiency change over time.*

Remark 1 *In our system, a server makes reputation by essentially burning resources (to generate a hashcash). Alternatively, a server can generate reputation by burning cryptocurrency. This way servers can use, for example, PoS-based solutions (which are more energy efficient than PoW-based solutions) to generate reputation. Note that a server must tie its proof-of-burn to*

its $serverID$ and $marketID$. This can be simply done by, for instance, paying to an output that requires $H(serverID||marketID)$ to be zero.

Remark 2 *We remark that anyone can join the market but may not necessarily profit from the market because of the competition that exists between servers to provide the service at the lowest possible fee. It is not our intention to guarantee that the market will have many servers in it. In fact, the market may become dominated by a small number of "powerful" servers. However, we aim at designing a market where every server has a strong incentive to fulfill its contracts, and lower its service fee in competing with other servers.*

Documents. The system generates two types of digital documents: contracts, and proof-of-breaches. A contract is a stand-alone digital document that includes a server ID, terms of the service, and the server's signature. A proof-of-breach, on the other hand, is a digital document that consists of a contract and publicly-verifiable evidence proving that the server who signed the contract has breached it.

Each contract contains two hash images: a *client hash image* selected by the client and a *server hash image* selected by the server. A contract is considered valid only if it is presented along with the preimage of the server's hash image. In contrast, a proof-of-breach becomes invalid if the preimage of the client's hash image is presented. As will be explained later, this mechanism allows a digital document to be atomically exchanged for cryptocurrency. One can view these preimages as one-time use on/off switches: the server's preimage activates the contract, while the client's preimage terminates it.

Distributed storage system. The reputation system utilizes a distributed storage system to store servers' records, including IDs, hashcash, and proof-of-breaches. The main requirement of this storage system is to ensure that each server's record is stored by *at least one node*, who is willing to share the record with others. As stated in the next proposition, this is a condition that is naturally achievable in our system. Consequently, the proposed reputation system can rely on the set of servers/clients as part of a (perhaps larger) distributed storage system that stores servers' records.

Proposition 3.1 *For every ID, hashcash, and proof-of-breach, there is at least one node in the system that has incentive to store and share the record.*

Proof. A server has full incentive to store and share its own ID and hashcash with others. With regards to a proof-of-breach, there is at least one node with a strong incentive to store and distribute the record: the victim of the contract breach. Note that the proof-of-breach has no expiry date. Therefore, the victim can store the proof for as long as they desire, and (re)distribute it anytime they wish. In addition to the victim, servers have incentives to store a proof-of-breach against the defaulting server. It is because the market share of the defaulting server becomes available to all others when the server is out of the market. Note that every server in the market essentially competes with every other server, because all servers offer their service to the same pool of clients. □

3.2 Interactions

The proposed reputation system supports two types of inter-component interactions: client-server interactions and client-storage interactions. Client-server interactions are to transfer reputation information including server ID, hashcash and server preimages (i.e., preimages that can invalidate existing proof-of-breaches). They are also to communicate terms of services and fees and to transfer contracts. Client-storage interactions, on the other hand, are to store and retrieve proof-of-breaches from the distributed storage system.

3.3 Protocol

Consider a market with a set of servers that provide a service for profit. Each server has an ID and a hashcash to represent its reputation. In this market, a client who wishes to receive the service goes through the following steps.

1. **Screening**: The client collects reputation information (including server IDs and hashcash) from the servers. In addition, it retrieves proof-of-

breaches from the distributed storage system and verifies them. Valid proof-of-breaches can be cached at the client side. The client discards servers with a valid proof-of-breach against them. The servers that are not discarded are referred to as *candidate servers*.

2. **Negotiation**: The client negotiates the terms of service and fees with the candidate servers through private communications. Alternatively, instead of actively engaging with each client, servers may offer fixed service plans that are ready to be signed.

3. **Selection**: Considering the servers' reputations and their service fees, a client selects a subset of the candidate servers to contract with. If the client selects more than one server, it will contract with each candidate server separately. We assume that the client receives no damage if at least one of the selected servers fulfills the terms of the contract. For example, in a watchtower market, if at least one watchtower respects its contract and monitors the blockchain, the payment channel is fully protected. If all the selected servers deny the fulfillment of the contract, however, the client can create a proof-of-breach against every single selected server.

   In this work, we do not impose any specific method for selecting the candidate servers. In fact, it may not be possible to enforce a fixed selection method, as clients may have other reasons (external to the system) to select a particular server. Nevertheless, we suggest the following properties to be considered in designing any selection method.

   (a) **Reputation-aware**: if the algorithm selects a server with reputation r and service fee f, then it must also select any server with reputation $r' > r$ and service fee $f' \leq f$;

   (b) **Fee-aware**: if the algorithm selects a server with reputation r and service fee f, then it must also select any server with service fee $f' < f$, and reputation $r' \geq r$.

(c) **Damage-aware**: for every selected server s, the contract's value for the client, $\text{val}(c)$, must be less than $\frac{\text{cost}(s)}{k}$, where $k \geq 1$ is a security parameter defined in Section 3.4.

The first two properties stimulate competition, and encourage servers to increase their reputation and reduce their service fees. Note that both reputation and service fee are considered by a client in selecting servers. The third property encourages servers to behave, and is a defence mechanism against bribery, as will be explained in Section 3.5.

4. **Purchase**: To purchase a contract, the client first obtains a signed copy of the contract from the server and verifies the contract. Then, the client purchases the contract by, for example, atomically exchanging the server's preimage for cryptocurrency. Recall that a signed contract is considered valid only when it is presented with this preimage.

5. **Punishment**: If the client ever discovers a breach in the contract, it creates a proof-of-breach and stores it in the distributed storage system. Optionally, the client can negotiate terms of settlements with the defaulting server. The contract supports the atomic exchange of the client's preimage (which invalidates the proof-of-breach) for cryptocurrency. In our model, a proof-of-breach against a server will reduce the server's reputation to zero, unless the server provides the corresponding client's preimage indicating that the contract has been terminated as a result of, for example, a settlement.

The main challenge in enabling the proposed reputation system is to enable publicly verifiable proof-of-breaches. We believe that many markets that provide blockchain services, such as timestamping or cryptocurrency exchange, can be (re)designed to provide this feature. In this work, we present one such service: blockchain monitoring.

3.4 Adversarial model

An adversary can join the market with an arbitrary number of IDs. Moreover, a client cannot determine if a set of IDs belong to the same entity. We assume that there are at least two independent service providers (servers) in the market[1].

An adversary can launch a denial of service attack by populating a storing node with server IDs and/or proof-of-breaches. It may also bribe a storing node to delete its proof-of-breach from its storage. An adversary can bribe a server to breach a contract. However, we assume that at each point in time, a server has standing bribes on at most k different contracts, where $k \geq 1$ is a system security parameter. In addition, we assume that the total amount of bribe offered to a server to breach a contract c is not more than the value of c for its client (otherwise, the bribe can be used to buy off the client directly!). Moreover, we assume that a server s does not accept any of its standing bribes, if the total amount of bribe offered to s (over all the contracts s is handling) is less than $\text{cost}(s)$, as defined in Definition 1.

For a single contract, a client may select and pay multiple servers. We assume that the client does not receive any damage if at least one of these servers fulfills the terms of the contract. For example, in the case of the watchtower market, the client receives no damage if at least one of the paid watchtowers monitors the blockchain and follows the terms of the contract (e.g., submit the justice transaction in case of cheating). In selecting servers, we assume that a client has a good estimate of the reputation cost function. Finally, we assume that the client can create a proof-of-breach against all paid servers if there is a term in the contract that is not fulfilled by any of the paid servers.

[1]In an open market, if the market is profitable for a single provider, it makes sense for another service provider to join the market.

3.5 Security Analysis

Sybil attack. In the Sybil attack, an adversary attempts to subvert the reputation system by creating multiple IDs. In our reputation system, an adversary cannot impact the reputation of servers by merely creating many IDs. It is because, unlike other reputation systems such as recommendation systems, in our system, the reputation of a server is not effected by other nodes. In fact, there are only two things that can impact one's reputation: proof-of-work and proof-of-breach.

DoS attack. An adversary may create multiple IDs and/or fake contracts to use up the storage of storing nodes. For instance, suppose that there are nodes in the system that provide clients with server IDs, hashcash, and their IP addresses. An adversary may try to overwhelm these nodes by flooding them with server IDs. At some point, a node does not have enough storage room to accept new IDs or has to replace old IDs with new ones coming.

A simple counter-measure against this type of denial-of-service attacks is to use servers' reputations to prioritize records. For example, a proof-of-breach against a highly-reputable server has a higher priority than a proof-of-breach against a normal server. With such prioritising, a storing node accepts and stores a new record if either the node has enough room or the priority of the new record is higher than the priority of the lowest-priority record in the storage. In the latter case, the lowest-priority record is replaced with the new record.

By the following proposition and considering today's computational power, storage capacity, and electricity cost it is impractical for an adversary to flush out the records of all honest servers from a storage node.

Proposition 3.2 *An adversary requires on average $M \cdot 2^{r_{max}}$ hash computations to flush out the records of all honest servers from a storage node, where M is the number of records the node can store, and r_{max} denotes the maximum reputation of any honest server.*

Proof. For an adversary to flush out the records of all honest servers from the

storage, it needs to create M records, each with reputation of at least r_{max}. To create a reputation of at least r_{max}, the adversary needs to compute on average $2^{r_{max}}$ hashes. Therefore, in total, the adversary needs to compute at least $M \cdot 2^{r_{max}}$ hashes, on overage. □

For example, assuming that $r_{max} \geq 43$, and the storage node can store 2^{40} records, the adversary must compute at least 2^{83} hashes in order to remove/replace all the honest servers' records. Using an ASIC hardware with the speed of 10 tera hashes per second this would take over 30,000 years. The same hardware can generate a reputation of $r = 43$ in less than a second, hence $r_{max} \geq 43$ is a reasonable assumption.

Proposition 3.3 *A defaulting server has no incentive to flush a proof-of-breach against itself out of a storing node by generating a new ID and many artificially-generated proof-of-breaches against the new ID.*

Proof. Suppose that the new artificially generated proof-of-breaches pushes out the valid proof-of-breach out of a storage node. Since the storage node prioritizes records according to the reputation of the corresponding server ID, the reputation of the new ID must be at least equal to the reputation of the defaulting server. There is no incentive then for the defaulting server to try to push out the proof-of-breach rather than starting fresh with the new ID. □

A storage node needs to verify records such as proof-of-breaches. Such verification needs little but non-negligible resources such as computation and memory. An attacker can overwhelm a storage node by sending many fake proof-of-breaches to the node. A storage node can mitigate this type of DoS attacks by simply requiring a small proof-of-work along with a submitted record. This mitigation is, in fact, the original application of the hashcash, which is to mitigate such DoS attacks.

Fake resolved proof-of-breaches. A server may create valid proof-of-breaches against itself, and then resolve them to create a market image of settling all disputes with clients. This method, however, does not impact the

server's reputation because terminated/resolved contracts neither increase nor decrease the server's reputation.

Eclipse attack. An attacker can eclipse honest servers to prevent (new) clients from discovering them. To combat this, servers can register their record on a secure blockchain. Such a record should include the server ID, reputation, IP address, and a signature to authenticate the record. Other methods such as those mentioned in [80] may also be used to mitigate the eclipse attack.

Bribery. A defaulting server may attempt to bribe the storing nodes to delete a proof-of-breach. Bribing, however, cannot guarantee that a digital document is purged. In fact, the victim of a contract breach would always keep its proof-of-breach and can re-distribute it at any time. The main hope of a defaulting server who is willing to remain in the market is to settle with the client (by purchasing the client's preimage, which invalidates the contract) or start over with a new reputation.

Another type of bribery is when an adversary offers a bribe to a server to breach a contract. For instance, a payment channel party may bribe a watchtower to stop monitoring the channel for its counter-party. This type of bribery is a serious threat against any digital market. Note that since a server provides services to many clients, it may receive multiple bribes from multiple adversaries. Nevertheless, Proposition 3.4 shows that every contract in the market is "bribe-safe", as defined below.

Definition 2 *(Bribe-safe) A contract is bribe-safe if the cost of every server responsible to execute the contract is more than the total amount of bribes offered to the server.*

Note that the condition in the above definition is somewhat strong, because a contract is still safe even when a single paid server (as opposed to all) fulfills its contractual obligation.

Proposition 3.4 *Under the assumptions described in the adversarial model, every contract in the proposed system is bribe-safe if the selection methods used by the clients are* damage-aware.

Proof. Let s be any server in the market, and $\mathcal{C}$ be the set of contracts for which the server has a standing bribe offer. Let $\text{brb}(c)$ denote the total amount of bribe offered to breach a contract c, $\text{val}(c)$ denote the maximum value of c for the client, and $\text{cost}(s)$ denote the cost of server s as defined in Definition 1. By the selection method's third property, we have

$$\forall c \in \mathcal{C}: \quad \text{val}(c) < \frac{\text{cost}(s)}{k}.$$

In addition, by our adversarial model assumptions, we have

$$\forall c \in \mathcal{C}: \quad \text{brb}(c) < \text{val}(c),$$

and $|\mathcal{C}| \leq k$, where $|\mathcal{C}|$ denotes the cardinality of $\mathcal{C}$. Therefore, we get

$$\begin{aligned} B &\leq \sum_{c \in \mathcal{C}} \text{brb}(c) < \sum_{c \in \mathcal{C}} \text{val}(c) \\ &\leq \sum_{c \in \mathcal{C}} \frac{\text{cost}(s)}{k} = \frac{|\mathcal{C}|}{k} \text{cost}(s) \leq \text{cost}(s), \end{aligned}$$

where B is the total amount of bribe offered to s. Thus, the total amount of bribe offered to s is less than $\text{cost}(s)$. Therefore, by our adversarial model's assumption, the server does not accept a bribe to breach any contract $c \in \mathcal{C}$. □

3.6 A reputation-based market of watchtowers

Our reputation system can be used in various digital markets which offer blockchain services. In this section, we show one example where we apply our system in a market of watchtowers that provides blockchain monitoring service to payment channel holders who decide to go offline.

Consider a market of watchtower servers, where each server has an ID (public key), and a reputation as defined in Section 3.1. The market utilizes a distributed storage system, which stores the server's records, including ID, hashcash, IP address, proof-of-breaches, and preimages. Recall that servers and victims of contract breaches participate in this distributed storage system,

and for each record, there is at least one node that has a strong incentive to store the record and distribute it.

Screening. Consider a payment channel between Alice and Bob and suppose that Alice is interested to pay one or more watchtowers to monitor the blockchain on her behalf while she is offline. First, in a screening process, Alice contacts the storage system, collects all servers' records, and evaluates servers. A server's evaluation includes a single computation of the hash function to calculate the server's reputation, and verification of proof-of-breaches against the server if there is any. Verification of a proof-of-breach is expected to be harder than a single computation of the hash function. However, a client needs to verify a proof-of-breach at most once, as it can cache the result for later use. The watchtowers that successfully pass Alice's evaluation are referred to as candidate watchtowers.

Negotiation. In this step, Alice communicates the terms of a contract with the candidate servers, and negotiates for service fees. At the end of this step, Alice knows how much fee each candidate watchtower charges for the contract. Figure 3.2 shows a watchtower contract sample, which consists of a market ID, server ID, server's hashcash, a set of 16-byte transaction ID prefixes, a set of encoded justice transactions, the range of blocks that the watchtower has to monitor, server's hash image, client's hash image, and server's signature. The client's and server's hash images are hashes of random numbers generated by, respectively, the client and the server. Note that contract's data does not reveal any information about the client.

Selection. Considering the reputations of the candidate watchtowers and their service fees, Alice selects a set of watchtowers to contract with. As mentioned earlier, we do not enforce any specific selection algorithm. We, however, suggest any selection method to be 1) reputation-aware; 2) fee-aware, and 3) damage-aware. Algorithm 1 shows one possible selection method. In this method, Alice first determines the contract's maximum value, $\text{val}(c)$. This value should be set to at least the payment channel fund, which is the maximum amount Alice would lose on the channel if Bob cheats. Then, among candidate watchtowers whose reputation cost is more than the thresh-

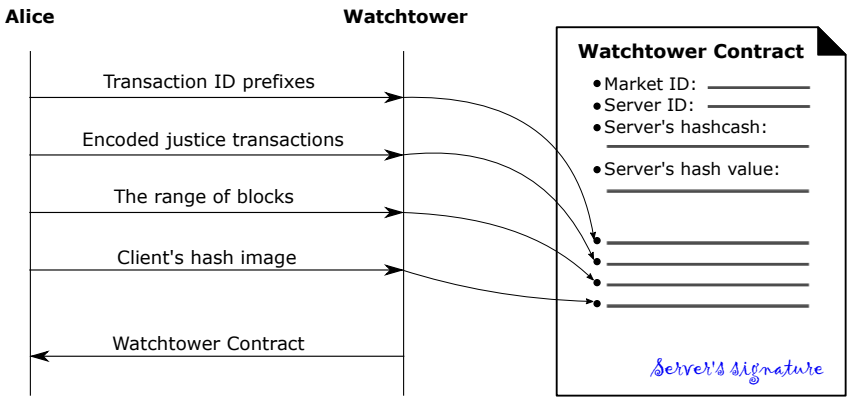


Figure 3.2: A sample of watchtower contract.

old $T = k \cdot \text{val}(c)$, the algorithm chooses the one with the minimum fee. If there are multiple such watchtowers with the minimum fee, one with the maximum reputation is selected. Thus, if the algorithm selects a server s with reputation r, and service fee f, then for every server s' we have

$$f' \geq f \quad \text{OR} \quad r' \leq r,$$

where f' and r' denote the service fee and reputation of s', respectively. This implies that the algorithm is both reputation-aware and fee-aware. In addition, the algorithm is damage-aware as the reputation cost of the selected server is at least k times the contract's value.

When selection is complete, Alice contacts the selected servers and provides them with her hash images[2]. In response, every server returns a signed contract. Note that a signed contract is only considered valid when it is presented with the server's preimage. To finalize the contract, Alice pays each server to receive the server's preimage.

Purchase. To purchase the server's preimage, Alice can use the Lightning network, or a payment channel with the watchtower. This may seem paradoxical, as the payment channel that Alice uses to pay the watchtower has to be protected by a watchtower, too. To get around this issue, Alice uses a directional payment channel to pay the watchtower. As explained in Section 2.6, Alice does not need to monitor the blockchain for a directional payment channel, i.e., a channel that she uses only to make payments.

[2]Different hash images are used for different servers.

Algorithm 1: A watchtower selection method.

```
Data: A threshold T, and a set of servers S
Result: s† ∈ S
initialization;
s† ← servers in S with reputation cost of at least T;
for every s ∈ S do
    if (cost(s) ≥ T) & (fee(s) ≤ fee(s†)) then
        if (cost(s) > cost(s†)) || (fee(s) < fee(s†)) then
            s† ← s;
        end
    end
end
```

To purchase the contract using a directional payment channel, Alice uses the watchtower's hash image in her HTLC. This, as discussed earlier, ensures that the contract and the service fee are atomically exchanged. As soon as the payment goes through (i.e., once Alice receives the watchtower's preimage), the contract becomes valid, and Alice can go offline.

Note that every time Alice transacts with Bob, she needs to update her contract with the watchtower (by updating a few parameters in the existing contract), and pay the watchtower to get a new preimage.

Proof-of-breach. Since the Bitcoin blockchain is public, anyone can check if, for example, the ctx transaction has been published. Therefore, anyone can verify whether a given contract has been breached. In particular, a contract together with the server's preimage serve as a proof-of-breach. Given a contract and a preimage, one can check whether

- the format of the proof-of-breach is valid;
- the contract's signature is verified using the server's ID;
- the hash image of the given preimage is equal to the server's hash image in the contract;
- ctx was published in one of the blocks that the watchtower was obliged to monitor;
- jtx is a valid transaction (e.g., it can spend from ctx);

- jtx was not published within the dispute period.

The proof-of-breach is valid if and only if all the above conditions hold. Supplementary data such as the Merkle proof of the existence of ctx can be appended to a proof-of-breach to assist light clients (e.g., app-based mobile clients) with the verification process[3]. This reduces the verification process to a few signature verification (e.g., checking the server's signature), a few hash computations (e.g., to verify a Merkle proof), and some simple format checking (e.g., checking the format of the contract).

Settlement. The victim of a contract breach may settle with the defaulting server. This can be done readily by atomically exchanging the client's preimage for cryptocurrency.

3.7 A reputation-based market of Timestamping

In the previous section, we showed how our reputation system can be used in a watchtower market to incentives watchtowers to respect their contracts, and compete with each other. Our reputation system can be adapted by other markets, too. The main challenged in doing so is to design contracts in such a way that a publicly verifiable proof-of-breach can be generated by a client if the server breaches the contract. In this section, we use a timestamping market as another example on how our reputation system can be used.

Consider a market in which servers provide blockchain-based timestamps for client's documents. As before, each server has an ID (public key), and a reputation as defined in Section 3.1. In this market, a server 1) receives and collects hash images of clients' documents; 2) creates a Merkle tree using the collected clients' hash images; 3) embeds the root of the Merkle tree in a transaction; and 4) publishes the transaction on the blockchain. When the transaction is published, the server can provide each client with a Merkle-proof to prove that the client's hash image was included in the Merkle tree.

[3]To show that jtx does not exist, the Merkle proof of the existence of a transaction (other than jtx) that spends from ctx can be provided. This proves that jtx does not exist because at most one transaction can spend the output of ctx.

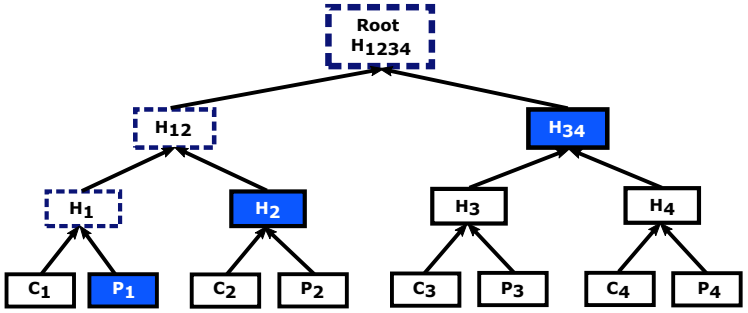


Figure 3.3: An illustration of how a timestamp server creates a Merkle tree.

The Merkle proof (together with the published transaction's ID) serves as a timestamp on the client's document. This timestamp proves that the client's document existed at a certain date (i.e., the date the transaction was published).

Why using a timestamp server? A client, say Alice, can timestamp her document on her own by embedding the document's hash image in a transaction and publishing it on the blockchain. This requires Alice to pay a transaction fee. Using the timestamp server, however, Alice pays much less for a timestamp, as a timestamp server's fee is much lower than a transaction fee; a server can afford this because it aggregates many clients' hash images into a single hash image (i.e., the root of the Merkle tree), and publishes this single hash image.

The second main motivation to use a timestamp server is to reduce the load on the blockchain. If every client uses their own transaction to timestamp their document, many transactions need to be published on the blockchain, which results in higher transaction fees.

Merkle tree and proofs. A Merkle tree [22] is a binary tree in which every non-leaf node is labelled with the cryptographic hash of the concatenation of its children. As shown in Figure 3.3, the timestamp server creates a Merkle tree using the client's hash images as the labels of every other leaf nodes. The remaining leaf nodes receive a secret unique preimage as a label. As will be explained later, these preimages are used by the server to sell Merkle proofs.

A Merkle proof is a proof that a leaf is a part of the tree with a given root. The Merkle proof for a leaf node consists of the labels of the siblings of

all of the nodes in the tree in the path from the leaf to the root. All these labels, except that of the leaf's sibling, is placed in the contract as the "partial Merkle proof". The label of the leaf's sibling is used as a preimage in the HTLC contract that sells the Merkle proof.

Screening, negotiation and selection. These steps are performed similar to the watchtower market. The only difference is in the set of items that are placed in the contracts.

As in a watchtower contract, a timestamp contract includes a market ID, server ID, the client hash image[4], the server's hashcash, and its signature on the whole contract. As shown in Figure 3.4, other items placed in a timestamp contract are:

1. the hash image of the client's document;
2. the ID of the transaction that includes the root of the Merkle tree;
3. the partial Merkle proof;
4. the server hash image, which is set to the hash of the label of the client's sibling in the Merkle tree.

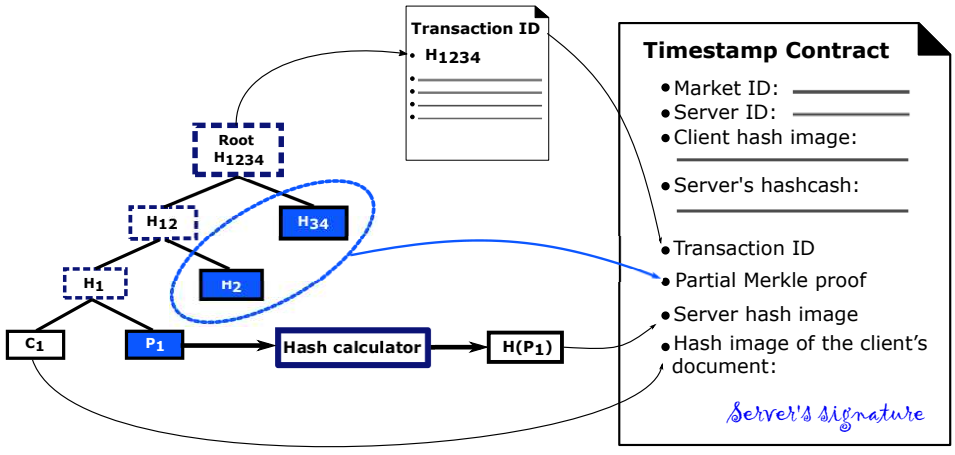
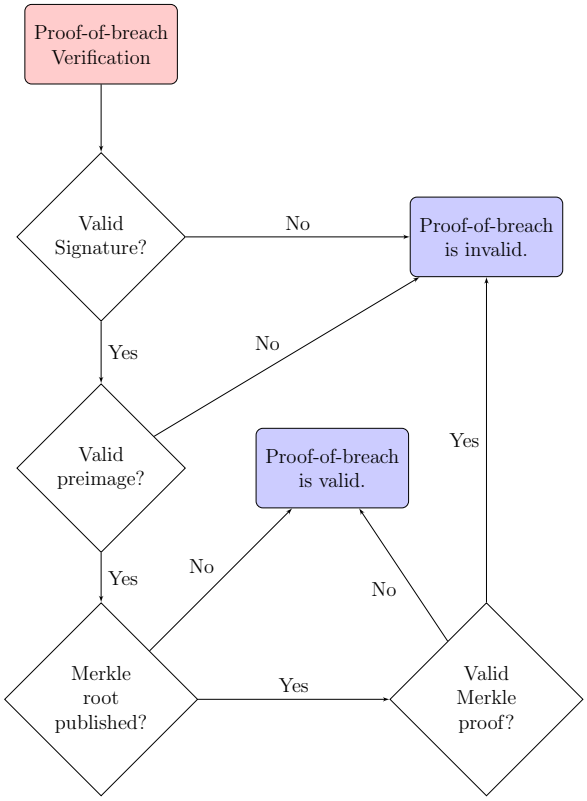


Figure 3.4: A sample of timestamp contract

[4]Note that this is different from the hash image of the client's document. Recall that the client hash image is used as a mean to deactivate the contract after a settlement.

Figure 3.5: The process of verifying a Proof-of-breach.



Purchase. The server's preimage is the only part missing from the Merkle proof (the server places the rest of the proof in the contract). To have the complete Merkle proof, the client has to purchase the server's preimage. As in the watchtower market, the server can atomically exchange its preimage for a fee using the Lightning network or payment channels. An issue, however, is that a server may give an invalid Merkle proof to the client. A client cannot verify the Merkle proof prior to getting the whole proof (which includes the server's preimage), and by the time the client has the whole proof, the exchange has already been completed (i.e., the server has already received the fee). This is the part that our reputation system comes to play; after paying the fee, if a client realizes that the Merkle-proof is invalid, she can create a publicly verifiable proof-of-breach, which destroys the server's reputation.

Proof-of-breach. As in the case of watchtower market, the contract together with the server's preimage that the client receives after the payment of the fee can be used as a proof-of-breach. To verify a proof-of-breach, one

should check whether

- the contract's signature is verified using the server's ID;
- the hash image of the preimage is equal to the the server hash image embedded in the contract;
- the root of the Merkle tree is in a published transaction, whose ID is given in the contract;
- the Merkle proof is valid.

The flowchart 3.5 shows how the answers to the above questions determines if the proof-of-breach is valid.

3.8 Conclusion

In this chapter, we proposed a proof-of-work based reputation system to incentivize well-behaviour and stimulate competition in online marketplaces. An advantage of our system is that it does not rely on any blockchain or smart contracts to, for example, punish misbehaviour. Instead, it stores a proof of misbehaviour as a record in a distributed storage, which is only required to store each record in at least one node. This is an easy requirement to achieve since for each record, there is at least one party who is strongly motivated to store and (re)distribute it. Finally, to showcase our reputation system, we designed an open market of watchtowers. Our reputation system motivates watchtowers not only to behave according to their obligation but also compete with each other by progressively improving their reputation and by reducing their service fees.

Chapter 4

Spear: Fast Multi-Path Payment with Redundancy

This chapter presents Spear, our simple multi-path payment method with redundancy. There is only one other method (called Boomerang) in the literature that supports redundant payments. In this chapter, we show that Spear has lower latency and computation than Boomerang and, unlike Boomerang, needs only a minor change to the Lightning Network. In addition, Spear trivially supports division of a payment into uneven partial payments. This gives the payer maximum flexibility to decide on the number of partial payments and their values.

4.1 System Model

We model the Lighting Network as a graph $G = (V, E)$, where V is the set of nodes and E is the set of channels. For any channel $(u, v) \in E$, let $b(u, v)$ denote the balance of node u, and $f(u, v, m)$ denote the fee that node u charges to forward a payment of amount m to v on channel (u, v). For each channel $(u, v) \in E$, the channel capacity is defined as $c(u, v) = b(u, v) + b(v, u)$.

Single-path payment. Let $\mathcal{P} = (v_1, v_2, \ldots v_l, v_{l+1})$ be a path in G. A single-path payment of m from v_1 to v_{l+1} on path $\mathcal{P}$ is called *successful* if and only if

$$\forall 1 \leq i \leq l : \quad b(v_i, v_{i+1}) \geq m_i, \tag{4.1}$$

where $m_l = m$, and m_i, $i < l$, can be calculated recursively as

$$\forall 1 \leq i < l : \quad m_i = m_{i+1} + f(v_i, v_{i+1}, m_{i+1}).$$

A successful transfer of payment m over $\mathcal{P}$ will result in the following changes in balances

$$\begin{aligned} \forall 1 \leq i \leq l : \quad b(v_i, v_{i+1}) &\longleftarrow b(v_i, v_{i+1}) - m_i \\ b(v_{i+1}, v_i) &\longleftarrow b(v_{i+1}, v_i) + m_i \end{aligned} \tag{4.2}$$

Multi-path payment. Another option for v_1 to transfer m to v_{l+1} is to split m into partial payments $m_1, m_2, \ldots, m_k$, where $\sum_{i=1}^{k} m_i = m$, and transfer m_i, $1 \leq i \leq k$, on a path $\mathcal{P}_i$. Such multi-path payment is called *successful* if and only if sequentially transferring of m_i on $\mathcal{P}_i$ result in k successful transfers according to (4.1). Note that channel balances are updated according to (4.2) after a successful transfer of a partial payment and prior to the transfer of the next partial payment.

4.2 Power of Redundancy

In [11], the authors experimentally showed that Boomerang, a multi-path payment with redundancy, can lead to 40% reduction in latency and 200% increase in throughput. In this section, we look at redundancy from a slightly different angle, and show its power in improving the success probability of payments.

It may seem at first glance that dividing a payment into partial payments and transferring them through different paths can significantly improve the chance of a successful payment. In this section, we show that this may not be the case if we do not add redundant partial payments. To this end, let us simplify our model by making the following assumptions

1. All channels have the same capacity C;
2. There is no service fee;
3. The balance of a channel is drawn uniformly at random independent of other channels.

With the above assumptions, we get the following proposition.

Proposition 4.1 *Let $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$ be k paths from node A to node B. Let l_i denote the length of path $\mathcal{P}_i$ (i.e., l_i is the number of channels on $\mathcal{P}i$). Let P_k denote the probability that a payment of $m = \sum_{i=1}^{k} m_i$ can be transferred from A to B by simultaneously transferring partial payments of m_i on path $\mathcal{P}_i$. Then, we have*

$$P_k = \Pi_{i=1}^{k} \left(1 - \frac{m_i}{C}\right)^{l_i},$$

if paths $\mathcal{P}_i$ are disjoint, and

$$P_k < \Pi_{i=1}^{k} \left(1 - \frac{m_i}{C}\right)^{l_i},$$

otherwise.

Proof. Suppose paths $\mathcal{P}_i$, $1 \leq i \leq k$, are disjoint. Let $\mathcal{E}_i$ be the event that the transfer of partial payment m_i on $\mathcal{P}_i$ is successful. We have

$$Pr(\mathcal{E}_i) = \left(1 - \frac{m_i}{C}\right)^{l_i},$$

where $\left(1 - \frac{m_i}{C}\right)$ is the probability that a channel has enough balance to forward m_i. The events $\mathcal{E}_i$, $1 \leq i \leq k$, are independent, because paths $\mathcal{P}_i$, $1 \leq i \leq k$, are disjoint. Therefore, we get

$$P_K = \Pi_{i=1}^{k} Pr\left(\mathcal{E}_i\right) = \Pi_{i=1}^{k} \left(1 - \frac{m_i}{C}\right)^{l_i}.$$

Now, let us consider the case where paths $\mathcal{P}_i$ are not disjoint. Consider any channel. The probability that this channel can transfer two partial payments m_i and m_j is

$$\left(1 - \frac{m_i + m_j}{C}\right)$$

which is less than

$$\left(1 - \frac{m_i}{C}\right) \cdot \left(1 - \frac{m_j}{C}\right).$$

This implies that transferring m_i and m_j on two different channels is more likely to succeed than transferring them on a single channel. Consequently, the success probability of transfer of payments is maximized when paths $\mathcal{P}_i$ are disjoint. □

Corollary 4.2 *Let l denote the length of the shortest path between A and B. Then, the probability that a payment m goes through a shortest path is*

$$P_{success}^{single} = \left(1 - \frac{m}{C}\right)^l.$$

Lemma 4.3 *Let l denote the length of the shortest path between A and B. Then we have*

$$P_{success}^{mult} < e^{-\frac{ml}{C}},$$

where $P_{success}^{mult}$ denotes the success probability of multi-path payment.

Proof. For any real number $0 < \alpha < 1$, and any integer $n \geq 1$, we have

$$(1-\alpha)^n < e^{-\alpha n}.$$

Therefore, by Proposition 4.1, we get

$$\begin{aligned} P_{success}^{mult} &= \max_{k,m_i,l_i} \left\{ \Pi_{i=1}^k \left(1 - \frac{m_i}{C}\right)^{l_i} \right\} \\ &< \max_{k,m_i,l_i} \left\{ \Pi_{i=1}^k e^{-\frac{m_i l_i}{C}} \right\} \\ &\leq \max_{k,m_i} \left\{ \Pi_{i=1}^k e^{-\frac{m_i l}{C}} \right\} \\ &= \max_{k,m_i} \left\{ e^{-\frac{\left(\sum_{i=1}^k m_i\right) l}{C}} \right\} \\ &= e^{-\frac{ml}{C}} \end{aligned}$$

□

Proposition 4.4 *We have*

$$P_{success}^{single} > 1 + \ln\left(P_{success}^{mult}\right)$$

Proof. By Corollary 4.2, the success probability of single-path payment is $\left(1 - \frac{m}{C}\right)^l$. Therefore

$$\begin{aligned} P_{success}^{single} &= \left(1 - \frac{m}{C}\right)^l \\ &\geq 1 - \frac{ml}{C} \\ &> 1 + \ln\left(P_{success}^{mult}\right), \end{aligned}$$

where the second inequality is by Lemma 4.3, and the first inequality is by the fact that $(1-\alpha)^n \geq 1-\alpha n$ for any real number $0 < \alpha < 1$, and any integer $n \geq 1$. $\square$

Example 1 *An important consequence of Proposition 4.4 is that if multi-path payment has a high probability of success so does single-path payment. For example, if* $P_{success}^{mult} = 99\%$*, then by Proposition 4.4, we get that* $P_{success}^{single} > 98.99\%$*. As another example, if* $P_{success}^{mult} = 90\%$*, then by Proposition 4.4, we get that* $P_{success}^{single} > 89.46\%$*. Note that this result holds for any values of* m*,* l*, and* C*.*

Proposition 4.4 essentially shows that if single-path payment has low probability of success, we cannot expect multi-path payment to achieve a high probability of success. By adding redundant payments to our multi-path payment, however, we can achieve a high probability of success even when the success probability of single-path payment is low. Let us clarify this in the following example.

Example 2 *Suppose* $l = 4$*,* $\frac{m}{C} = \frac{1}{3}$*, and there are* $k = 10$ *disjoint paths. Then, by Proposition 4.1, we get*

$$P_{success}^{single} = \left(1 - \frac{m}{C}\right)^l = \left(1 - \frac{1}{3}\right)^4 \approx 20\%$$

and

$$P_{success}^{mult} = \left(1 - \frac{m}{kC}\right)^{kl} = \left(1 - \frac{1}{10 \times 3}\right)^{10 \times 4} \approx 26\%$$

Now, if we use a multi-path payment method with redundant payments by transferring $\frac{m}{5}$ *on every path, then the probability that at least five of these partial payments are successfully received can be shown to be at least* 98%*. This means that the probability of success of multi-path payment with redundancy can be higher than* 98%*, that is*

$$P_{success}^{redundant} > 98\%,$$

where $P_{success}^{redundant}$ denotes the success probability of multi-payment with redundancy. Note that by Lemma 4.3, the probability of success of multi-path payment (without redundancy) for this example is at most

$$P_{success}^{mult} < e^{-\frac{ml}{C}} = e^{-\frac{4}{3}} \approx 26.36\%,$$

even when there are infinitely many disjoint paths.

4.3 Spear

An overview. Suppose Alice wishes to send a payment to Bob over a single path. To do so, Alice first acquires a hash digest from Bob. Then, she selects a single path to Bob, and sends the payment (conditioned on Bob disclosing the preimage) through the selected path. Finally, Bob accepts the payment by releasing his preimage. In this single-payment scheme, Bob's hash digest can be used as part of an invoice, and its preimage can be treated as a receipt/proof of payment.

Spear can be viewed as an extension of the above single-payment scheme. It allows Alice to select multiple paths, including some redundant ones, to send partial payments to Bob. Spear follows the same procedure as the single-payment scheme, except it uses a slightly modified version of Hashed Timelock Contract (HTLC), which is the contract that conditions the release of payment on the disclosure of Bob's preimage. As will be explained later, the main purpose of using the new HTLC is to prevent Bob from overdrawing. We refer to this new HTLC as H^2TLC.

Figure 4.1 compares HTLC and H^2TLC. As reflected in the figure, the only difference between HTLC and H^2TLC is that H^2TLC uses two hash digests instead of one. In other words, the transfer of money in H^2TLC is conditioned on releasing two preimages instead of one. In Spear, one hash digest is set by Bob, while the second one is set by Alice. This simple addition gives both parties control over the release of the payment in the H^2TLC contract.

As in the single-payment scheme, in Spear Alice first acquires a hash digest from Bob. She then uses this hash digest in setting up H^2TLC in every selected

```
HTLC:{
  Vout: [{
    value: <payment>
    scriptPubKey:
      IF
        HASH160 <Hash_Bob>
        EQUALVERIFY <PK1>
      ELSE
        <delay> CSV DROP <PK2>
      ENDIF
      CHECKSIG
  }]
}
```

(a) HTLC

```
H2TLC:{
  Vout: [{
    value: <payment>
    scriptPubKey:
      IF
        HASH160 <Hash_Alice>
        EQUALVERIFY
        HASH160 <Hash_Bob>
        EQUALVERIFY <PK1>
      ELSE
        <delay> CSV DROP <PK2>
      ENDIF
      CHECKSIG
  }]
}
```

(b) H^{2}TLC

Figure 4.1: HTLC and H^{2}TLC in Bitcoin Script Pseudocode.

path. The second hash digest in a H^{2}TLC is, however, set by Alice alone. Unlike Bob's hash digest which is the same on every path, Alice's hash digest varies from one path to another. When Bob receives multiple partial payments whose sum is equal to the whole payment, he would contact Alice over an out-of-band channel and ask her to provide him with the corresponding preimages. Alice will check whether the sum of the partial payments is indeed equal to the original payment, and if so she sends the requested preimages to Bob. At this stage, Bob can accept all the partial payments by releasing his own preimage together with Alice's preimages. As in the case of single-payment scheme, Bob's preimage can be treated as a proof of payment.

4.3.1 Procedure

Spear makes a payment in four steps:

- **Step 1:** Alice receives an invoice from Bob through an out-of-band channel. The invoice includes the amount F that Alice has to pay and a hash digest h_b[1]. We remark that, unlike Boomerang, Alice and Bob are not

[1]Bob generates h_b by first drawing a secret (preimage), and then applying a cryptographic hash function to it.

required to agree on how F will be divided into partial payments; Spear allows Alice to divide F to any number of partial payments. Moreover, Spear allows Alice to set uneven partial payments, as explained in the next step.

- **Step 2:** Alice selects k payment paths to Bob. For each path, she sets an amount of partial payment, and a unique hash digest by applying a cryptographic hash function to a secret unique to the path. She then initiates the transfer of the partial payments all together. Note that, in Spear, Alice can choose any number of paths, and any amount for the partial payments. For example, suppose $F = \$8$. Alice can set k to 12 and send \$1 on each of these 12 paths. Or, she may select seven paths (i.e. $k = 7$), send \$4 on the first path and \$2 on each of the remaining six paths.

- **Step 3:** Using the out of band channel, Bob informs Alice of the partial payments he has received and request preimages to claim the payment. In response, Alice will reveal a subset of her preimages to Bob. To prevent Bob from overdrawing, Alice makes sure that the sum of the partial payments corresponding to the released preimages is equal to F.

- **Step 4:** Let $\mathcal{P}$ be the set of partial payments whose preimages have been released by Alice in Step 3. In Step 4, Bob claims all the partial payments in $\mathcal{P}$ if

  1. The sum of payments in $\mathcal{P}$ is at least F, and
  2. Bob has enough time to claim all the payments in $\mathcal{P}$.

  While claiming the partial payments, Bob cancels any received redundant payment.

Figure 4.2 illustrates Steps 2 and 4 of the above procedure. Note that these steps are basically equivalent to applying the conventional single-payment scheme multiple times, with the only exception that each partial payment uses H^2TLC instead of HTLC. In particular, notice that the timeouts on each

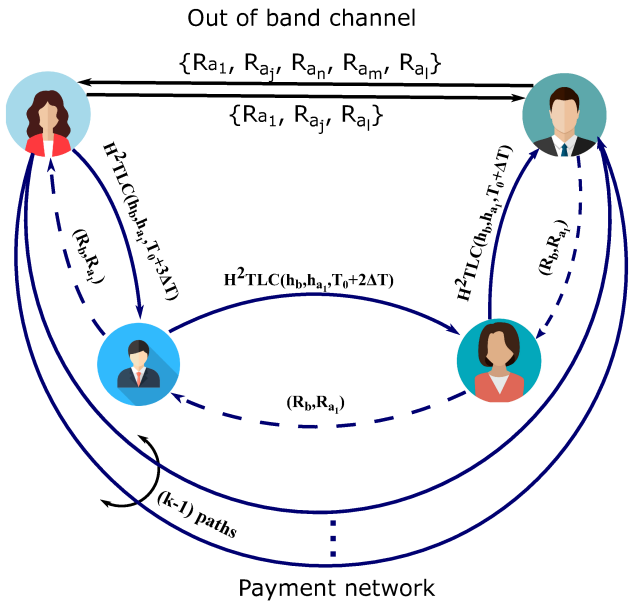


Figure 4.2: Making a payment in Spear.

path follows those of the single-payment scheme. Also, notice that Steps 1 and 3 each require only a single round of communication over an out-of-band channel.

4.3.2 Security Guarantees

As mentioned earlier, Spear prevents Bob from ever overdrawing. The following proposition captures this essential property of Spear.

Proposition 4.5 *Bob cannot overdraw if Alice follows the protocol.*

Proof. To claim a partial payment, Bob has to release two preimages. Since one of the two preimages is only known by Alice and is unique per partial payment, the only way Bob can claim a partial payment is to know Alice's preimage for the partial payment. Therefore, the maximum amount Bob can ever claim is limited to the sum of the partial payments whose preimages have been released by Alice in Step 3. This sum is equal to F according to Step 3 of the protocol. □

As mentioned earlier, Bob's secret preimage acts as a proof of payment, hence must remain secret until Bob receives the full payment. Spear guarantees

this as stated in the next proposition.

Proposition 4.6 *If Bob follows the protocol, then Alice will know Bob's secret preimage only if Bob receives the full payment.*

Proof. According to Step 4 of the Spear procedure, Bob does not claim any partial payments if he is not guaranteed to receive at least an amount of F. In other words, Bob will release his secret preimage only if he is guaranteed a total payment of at least F. □

The following corollary is a direct consequence of Propositions 4.5 and 4.6.

Corollary 4.7 *Alice can use Bob's secret preimage as a proof that she has paid Bob.*

4.3.3 Implementation

Implementing Spear is relatively simple. In Spear, each partial payment is handled similar to the conventional single-payment scheme with only one exception: the transfer of payment on channels must be conditioned on the release of two preimages instead of one. We emphasize that, in Spear, all other parameters such as timeouts remain the same as the conventional single-payment scheme.

Consider any channel on a partial payment path. Suppose this channel is between nodes C and D. According to BOLT[2][81], node C must send an `update-add-htlc` message to node D in order to create an HTLC. In Spear, however, node C needs to create an H^{2}TLC instead of an HTLC. For this, C must send an `update-add-H`^{2}`TLC` message to node D. As shown in Figure 4.3, `update-add-H`^{2}`TLC` is basically an `update-add-htlc` message with an additional field, which carries Alice's hash digest. The size of an `update-add-htlc` message is 1450 Bytes. Since `update-add-H`^{2}`TLC` carries an extra element (i.e., a hash digest of size 32 bytes) its size is 1482 bytes, which is about 2% larger than the size of an `update-add-htlc` message.

[2]BOLT (Basis of Lightning Technology) is the standardized technical specification for the implementation of the Lightning Network.

1. type: 128 (update_add_htlc)
2. data:
   - [channel_id:channel_id]
   - [u64:id]
   - [u64:amount_msat]
   - [sha256:payment_hash]
   - [u32:cltv_expiry]
   - [1366*byte:onion_routing_packet]

(a) HTLC

1. type: 128 (update_add_h^{2}tlc)
2. data:
   - [channel_id:channel_id]
   - [u64:id]
   - [u64:amount_msat]
   - [sha256:payment_hash]
   - [sha256:sender_payment_hash]
   - [u32:cltv_expiry]
   - [1366*byte:onion_routing_packet]

(b) H^{2}TLC

Figure 4.3: Upgraded update-add-htlc message. Refer to Figure 4.4 for the `Onion_routing_packet` field.

4.4 Spear versus Boomerang

In [11], the authors devise Boomerang, and show that the latency of transfers reduces and the throughput increases when redundant payment paths are added. Our work is motivated by this positive result and aims to improve it through the design of Spear. To evaluate the improvement, in the following, we compare Spear and Boomerang against several factors, including payment latency, implementation complexity, computational overhead, and required liquidity and timeouts.

4.4.1 Latency

The main objective of both Spear and Boomerang is to reduce the payment latency, that is the time needed to complete the payment process. Therefore, it is interesting to first see how these two methods compare to each other in

```
1. type: onion_packet
2. data:
    ◦ [byte: version]
    ◦ [point:public_key]
    ◦ [1300*byte:hop_payloads]
    ◦ [32*byte:hmac]
```

(a) onion packet

```
1. type: hop_payloads
2. data:
    ◦ [bigsize:length]
    ◦ [hop_payload_length:hop_payload]
    ◦ [32*byte:hmac]
    ◦ ...
    ◦ filler
```

(b) hop payloads structure

Figure 4.4: The `onion_routing_packet` field. Refer to Figure 4.5 for the `hop_payload` field.

terms of latency.

Both Spear and Boomerang use two types of exchanges: 1) exchanges over the Lightning Network, and 2) exchanges over an out-of-band channel. The main advantage of Spear over Boomerang with regards to latency is that it needs two exchanges of the former type while Boomerang requires three: both Spear and Boomerang use the first exchange to forward Alice's partial payments to Bob, and the second exchange for Bob to release his secret preimage. At the end of the second exchange, the transfer of payment is complete in Spear, while Boomerang requires an additional exchange so Alice can free up liquidity. As will be explained, exchanges over the Lightning Network are considerably slower than exchanges over an out-of-band channel. Therefore, one can expect Boomerang to be about 50% slower than Spear (as it requires three exchanges over the Lightning Network as opposed to two). In the following, we analyze this claim.

For a fair comparison, let us assume that both Spear and Boomerang use the same set of paths and the same amount of partial payments on these paths.

```
1. type: hop_data (for realm 0)
2. data:
    ◦[short_channel_id:short_channel_id]
    ◦[u64:amt_to_forward]
    ◦[u32:outgoing_cltv_value]
    ◦[12*byte:padding]
```

(a) hop data

```
1. tlv_stream: tlv_payload
2. types:
        i. type: 2 (amt_to_forward)
       ii. data:
            ▪[tu64:amt_to_forward]
      iii. type: 4 (outgoing_cltv_value)
       iv. data:
            ▪[tu32:outgoing_cltv_value]
        v. type: 6 (short_channel_id)
       vi. data:
            ▪[short_channel_id:short_channel_id]
      vii. type: 8 (payment_data)
     viii. data:
            ▪[32*byte:payment_secret]
            ▪[tu64:total_msat]
```

(b) tlv-payload

Figure 4.5: The `hop_payload` field.

Proposition 4.8 *Let δ denote the average round-trip time between two nodes that are connected with a channel, γ denote he average round-trip time between Alice and Bob, and l denote the length of the longest path over which a partial payment is transferred from Alice to Bob.*
Then, the average payment latency of Boomerang and Spear can be estimated as $6l \cdot \delta + \gamma$ and $4l \cdot \delta + 2\gamma$, respectively.

Proof. To estimate the latency of the two payment protocols, we break their process into sequential steps, and then estimate the time they need for each step. In the first step, both processes require a single round of communication between Alice and Bob over an out-of-band channel. Spear requires this step so Alice can obtain the hash digest from Bob, while Boomerang requires this

step so 1) Alice can obtain the polynomial coefficients from Bob, and 2) Alice and Bob can agree on the number of partial payments. By definition of γ, this step requires γ seconds.

In the second step, both protocols transfer partial payments from Alice to Bob. All these transfers occur in parallel, hence the time needed for this step can be estimated by the time needed for the partial payment to go through the longest path. Let l be the length of the longest path whose partial payment is accepted by Bob. The partial payment on this path goes through l channels, sequentially. At each channel, the two "channel holders" must create new commitment transactions [12]. To perform this, as illustrated in Figure 4.6, first the two users agree to make a new commitment. Then each user generates a new commitment transaction. Since the commitment transaction is revocable, users must sign and exchange Revocable Delivery Transaction (RDTX). In addition, they must cancel the previous commitment transaction by signing a Breach Remedy Transactions (BRTX), and exchanging these transactions. This process, as shown in Figure 4.7, requires two rounds of communications between the two channel holders, hence requires about 2δ seconds. Since this process is repeated sequentially over l channels, the second step in both Spear and Boomerang require about $2\delta \cdot l$ seconds[3].

In the next step, Spear requires a single round of communication between Alice and Bob over the out-of-band channel. In this step, Bob informs Alice about the set of receive partial payment. In response, Alice sends Bob the corresponding preimages. This step therefore needs γ seconds. Note that Boomerang does not require this step. In the next step, both Spear and Boomerang require Bob to release his secret and claim the partial payments. Similar to the second step, this step requires about $2\delta \cdot l$ seconds in both protocols. In the final step, which is only required by Boomerang, Alice renounces the option to react and frees up the liquidity. Similar to the previous step, this step requires $2\delta \cdot l$ seconds.

[3]In this step, Bob cancels the redundant partial payments at the same time that he accepts the other partial payments. The time needed to finish this step is, therefore, more accurately captured as $2\delta \cdot \max(l, l')$, where l' is the length of the longest redundant path.

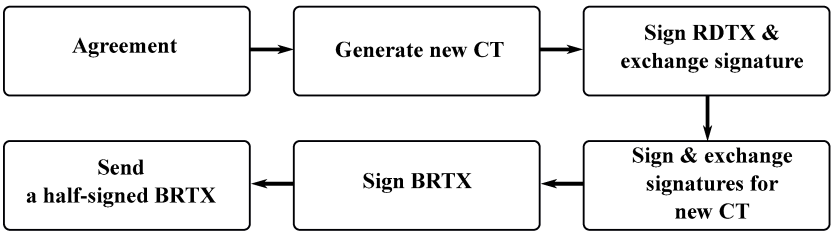


Figure 4.6: The process for creating new commitment transactions

Adding delays of all steps for the two protocols, we get that Spear requires about $4l \cdot \delta + 2\gamma$ seconds, and Boomerang needs about $6l \cdot \delta + \gamma$ seconds.

□

Corollary 4.9 *The payment latency of Boomerang is up to* 50% *higher than that of Spear.*

Proof. By Proposition 4.8, the ratio of the payment latency of Boomerang over the payment latency of Spear can be approximated as

$$\frac{6l \cdot \delta + \gamma}{4l \cdot \delta + 2\gamma},$$

which is at most equal to 1.5. □

Example 3 *Suppose Alice is paying Bob for a cup of coffee she is purchasing at Bob's coffee shop. Assuming that γ is much smaller than δ (e.g., Alice and Bob are connected to the same network), we get*

$$\frac{6l \cdot \delta + \gamma}{4l \cdot \delta + 2\gamma} \approx \frac{6l \cdot \delta}{4l \cdot \delta} = 1.5.$$

On the other hand, if the connection between Alice and Bob has a round-trip time of δ, then, assuming $l = 4$, we get

$$\frac{6l \cdot \delta + \gamma}{4l \cdot \delta + 2\gamma} = \frac{6l \cdot \delta + \delta}{4l \cdot \delta + 2\delta} = \frac{25\delta}{18\delta} \approx 1.39.$$

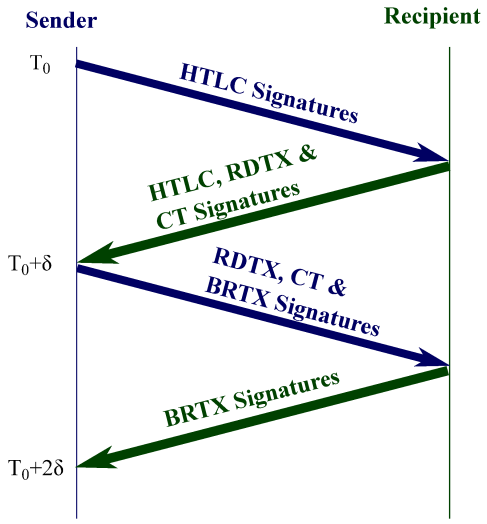


Figure 4.7: The process of creating an HTLC.

4.4.2 Implementation

Boomerang's implementation requires a new opcode or use of adaptor signatures [11]. Implementing Spear, on the other hand, is straightforward; we mainly need to replace HTLC with H^{2}TLC, which is identical to HTLC, except it conditions the payment on release of two preimages instead of one.

4.4.3 Locktime

As shown in Figure 4.8, for a payment path of length l (i.e., a path with l channels), Spear requires a maximum locktime of $l \cdot \Delta$, where Δ is the time set so that a node can forcefully redeem a fund on chain[4]. This is identical to the maximum locktime set in the conventional single-path payment scheme. Boomerang, on the other hands, requires the maximum locktime of $2 \cdot l \cdot \Delta$ [11], which is twice what is needed in Spear. Therefore, in the worst case (e.g., as a result of a griefing attack) funds can be locked for a longer period of time in Boomerang than in Spear.

[4]Values of Δ are typically either 40 blocks or 144 blocks [82].

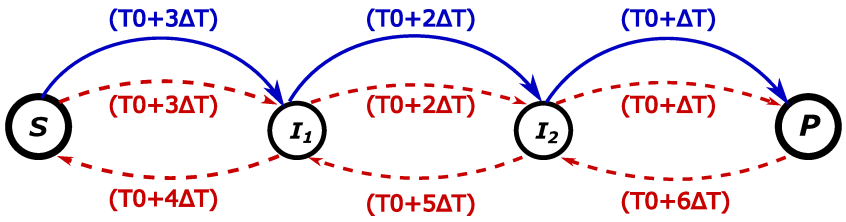


Figure 4.8: Transfer of a partial payment from S to P. The red dashed lines, and solid blue lines show locktime in Boomerang and Spear, respectively.

4.4.4 Computational overhead

The computation that Spear imposes on Bob is limited to computing a hash function per partial payment. In Boomerang, on the other hand, Bob has to compute a finite field exponentiation per partial payment, with each exponential requiring at least q field multiplications, where q is the order of the finite field in bits. Assuming that a field multiplication is of the same order of computation as a single hash computation, the computational overhead imposed by Boomerang is at least a factor of f of that of Spear[5]. For example, assuming that Boomerang uses a finite field of order 256 bits, we get that the computational overhead that Boomerang imposes on Bob is at least two orders of magnitude higher than what Spear imposes.

As for Alice, Boomerang requires at least v^2 field multiplications per partial payment, where v is the maximum number of partial payments that Bob can accept (Proposition 4.10). Spear, however, requires a single hash computation per partial payment. Therefore, Boomerang imposes more computational overhead on Alice than Spear does. In particular, note that Boomerang's required computation per partial payment grows quadratically with the number of partial payments, while in Spear, the amount of computation per partial payment is constant.

Proposition 4.10 *In Boomerang, Alice requires to compute at least v^2 field multiplications per partial payment.*

[5]This factor can be made smaller (in the best case to $\frac{f}{\log f}$) at the expense of more storage.

Proof. In Boomerang, Alice has to compute

$$\forall i \in \{1, 2, \ldots, v, \ldots, w\} \quad H(P_i) = \Pi_{j=0}^{v} H(\alpha_j)^{\binom{i}{j}}, \tag{4.3}$$

where w is the total number of partial payments, v is the maximum number of partial payments that Bob can accept, and $H(\alpha_j)$ are digests that Bob provides Alice. By (4.3), Alice needs to compute $w \cdot v$ exponentiations. Note that the largest exponent in (4.3) is w^v. Therefore, by Yao's result [83], Alice needs to perform at least

$$(w \cdot v + o(1)) \log(w^v) / \log\log(w^v) \sim w \cdot v^2$$

field multiplications. Therefore, the minimum number of field operations needed by Alice is $(w \cdot v^2)/w = v^2$. □

4.4.5 Flexibility

In Boomerang, Alice and Bob must agree (out-of-band) to partition the payment into v partial payments. In addition, Boomerang inherently requires all the partial payments to be of the same value. In Spear, on the other hand, Alice can decide on her own how to divide the payment into partial payments. Moreover, Spear allows Alice to divide the payment into unequal partial payments. This provides high flexibility to Alice to handle the payment. For instance, if Alice has a prior knowledge that a certain path (e.g. a direct channel to Bob) can carry a large portion of the total payment, then she can use this path to transfer a large partial payment to Bob in a single partial payment. Or, Alice may try to transfer larger partial payments on shorter paths than on larger paths, since a partial payment is more likely to go through a shorter path than a larger one (e.g., see Corollary 4.2).

4.4.6 Intermediate node's misbehaviour

In the Lightning Network, an intermediate node can stall a payment for hours by accepting an incoming payment and not forwarding the payment to the next node. In the single-path payment scheme, Alice has to wait for a long period

of time (until this payment is canceled) before she can make a second attempt to pay Bob. HTLC-based multi-path payment methods are also vulnerable; if a single partial payment is stalled by a misbehaving node, Alice has to wait until this partial payment is canceled before she can make a second attempt. This may not be desired, as Alice may prefer to pay Bob as soon as possible, even if one or more partial payments are stalled/locked for a period of time.

Multi-path payment methods with redundant paths such as Spear and Boomerang naturally mitigate this issue. It is because these methods, by definition, support redundant paths, and as long as Bob receives enough number of partial payments, the payment can go through. However, there is a difference between Spear and Boomerang when it comes to freeing up liquidity.

In Boomerang, if a misbehaving node stalls a single partial payment on path P then Alice may not free up the liquidity on other paths until the partial payment on P is canceled. It is because, Alice cannot distinguish between the following two scenarios: 1) an intermediate node on path P is stalling the partial payment; 2) Bob is holding on to the partial payment on P so he can claim it later. Therefore, Alice may not renounce the reverse components of Boomerang contracts to free up their liquidity, as there is a chance that Bob overdraws later, at which point Alice can revert the partial payments. In Spear, on the other hand, Bob can immediately free up the liquidity on all the paths except P by accepting the partial payments, and cancelling the redundant ones. Therefore, in Spear, the misbehaving node on path P can only delay the process of freeing up the liquidity of path P but not other paths.

4.4.7 Fees

In both Boomerang and Spear, the sum of partial payments Bob receives can be higher than the full payment. In this case, assuming Bob is honest, he will only claim a subset of the received partial payments whose sum is equal to the full payment. In Spear, Bob can go a bit further and help Alice to reduce her fees by reporting all the partial payments he receives. This allows Alice to choose the subset of partial payments whose sum of fees is minimum. In

Boomerang, on the other hand, Bob does not know which subset of partial payments to accept in order to reduce Alice's fees. It is because, Bob is unaware of the fees Alice is paying on each partial payment path (unless Alice communicates this information to Bob).

4.5 Conclusion

In this work, we showed that a payee can significantly improve the success probability of payments by sending redundant partial payments. We proposed Spear, a simple multi-path payment method with redundancy, and showed how it can be implemented in the Lightning Network. Moreover, we compared Spear with Boomerang, the only existing multi-path payment with redundancy in the literature. Our comparison results show that Spear has several advantages over Boomerang, including lower delay, ease of implementation, and lower computational requirement. The approach used in Spear can be used in other payment methods, such as packet-switch routing methods, for further performance improvement.

Chapter 5

Torrent: Balance Discovery in the Lightning Network Using Maximum Flow

In this chapter, we propose Torrent, a powerful probing technique to discover balances of channels in payment networks. Torrent can concurrently send several payments through multiple paths and consequently speeds up balance discovery of channels. Torrent uses a novel max flow algorithm in the Lightning network to discover balances of remote channels.

5.1 Torrent

Torrent can be viewed as a multi-path payment approach to channel discovery; rather than probing the balance of a target channel through a series of sequential single-path payments—as done in the existing methods—Torrent uses multiple paths to concurrently transmit partial payments through the target channel. To ensure that these partial payments are not canceled during the channel discovery process, receiver(s) in Torrent temporarily hold the received payments.

Let f_{max} denote the maximum flow that Torrent can push through the target channel assuming that the target channel has an infinite balance. Torrent works based on the idea that the balance of a target channel can be discovered if f_{max} is greater than the balance of the channel. This is because, under the above condition, the balance of the target channel becomes the bottleneck

for the maximum flow that can be transferred through the target channel. Under this condition, therefore, balance of the target channel is equal to the maximum amount of money that can be sent through the target channel.

In this work, we show this condition holds for a vast majority of channels in the Lightning Network.

Example 4 *Figure 5.1 shows a small payment networks with 11 channels and 10 nodes. Carol has two nodes and is interested to know the balance of Alice on the Alice-Bob channel. Assume that there are enough balances on Carol's channels. If we change Alice's balance from 200k to ∞, then the maximum flow that Carol's node on the left (the sender) can send to her node on the right (the receiver) is 230k Satoshis. Note that Carol can send*

$$\begin{aligned} 60k \quad & \textit{via path } (n_1, n_2, Alice, Bob, n_4, n_6) \\ 160k \quad & \textit{via path } (n_1, n_3, Alice, Bob, n_5, n_6) \\ 10k \quad & \textit{via path } (n_1, n_2, Alice, Bob, n_5, n_6) \end{aligned}$$

Also, note that cutting channels (Bob, n_4) and (n_5, n_6) will disconnect the sender and the receiver. These two channels can carry a total flow of

$$60k + 170k = 230k$$

Therefore, $f_{max} = 230k$, which is higher than Alice's balance of 200k. Consequently, in this example, Torrent is able to discover Alice's balance. We remark that the maximum flow that a single path from Carol's sender to her receiver can carry is at most 160k. Therefore, the best conclusion that the existing channel discovery methods can make in this example is that Alice's balance is at least 160k.

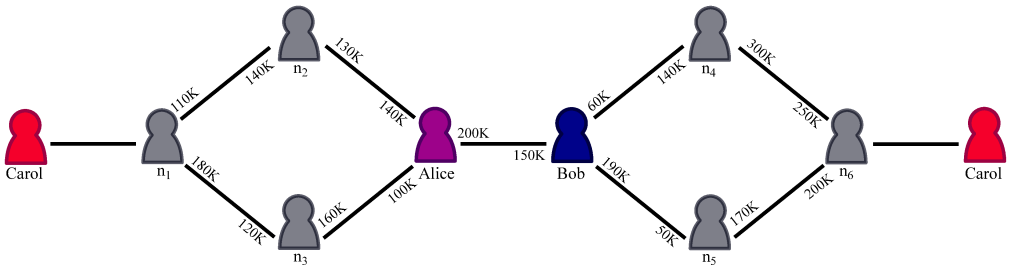


Figure 5.1: A small payment network. Carol has two channels with nodes n_1 and n_6, and is interested to discover balance of the channel between Alice and Bob. Balances of Alice and Bob on this channel are 200k and 150k Satoshis, respectively.

Types of Torrents. As mentioned earlier, Torrent requires the receiver(s) of payments to temporarily hold any received payment. To achieve this, Torrent can employ its own receiver(s). Clearly, in this case, receivers would follow Torrent and temporarily withhold received payments as needed. We refer to the case where Torrent uses its own receiver(s) as *Torrent Type A*.

Alternatively, in what we call *Type B*, Torrent uses Alice and/or Bob as receivers. This is possible if Alice and/or Bob support AMP [70], a multi-path payment method supported in the new versions of LND[1] [84]. Note that in AMP (as in any multi-path payment method) a receiver needs to hold partial payments until it receives the payment in full. Torrent can benefit from this property of AMP and use Alice and/or Bob as receivers instead of employing its own receiver(s).

Note that Torrent Type A not only needs to push a large flow through into the target channel, but also has to push this flow out of the target channel and deliver to its receiver(s). Torrent Type B, however, only needs to push a large flow towards the target channel. As a result, in general, Torrent Type B can push a larger flow than Type A. For instance, in the network of Figure 5.1, Carol can send a flow of size 110k+160k= 270k in Torrent Type B, while the maximum flow in Type A (as shown in Example 4) is 230k.

[1]About 90% of nodes in the Lightning Networks are LND nodes [82].

5.2 Implementation

As before, suppose Carol is interested to know Alice's balance. To implement Torrent, Carol needs to open a set of channels, and use them to send/receive a large flow of payments through the target channel, i.e. Alice's channel. If Carol knows balances of all channels except the target channel, then she can use linear programming—as we will show later—to find an optimal set of payment paths that can transfer the maximum flow through the target channel. In practice, however, Carol does not know balances of channels. This makes the problem of finding the max flow challenging.

To solve the above unique max flow problem, we propose two algorithms. The first algorithm assumes that all channels, except the target channel, have a certain minimum balance. For example, it may assume the balance of every channel is at least 10% of its capacity. Using this assumption, the algorithm uses a linear program to solve the max flow problem. The advantage of this approach is that it can discover balances very fast because 1) it can pre-compute paths and the amount of payments that should be transferred on each path; 2) it can send all the payments in parallel as the amount of payments are known a priori. Of course, the minimum balance assumption may not hold for all channels. However, this may not negatively impact the algorithm if the feasible max flow is considerably higher than the balance of the target channel. In addition, the algorithm can always re-attempt a failed payment by trying smaller payments on the failed paths.

The second algorithm is a greedy iterative algorithm. In each iteration, the algorithm updates its view of the network based on results of the previous iteration, then selects a set of short disjoint paths and pushes the maximum possible flow through these paths.

In the following, we explain these two algorithms for Torrent Type A. These algorithms can be easily converted and used for Type B, because Type B is essentially a special case of Type A where receivers are the owners of the target channel. We start by modeling the Lightning Network and defining our max flow problem.

5.2.1 Network Model and Problem Definition

We model the Lighting Network as a graph $G = (V, E)$, where V is the set of nodes and E is the set of channels. For any channel $(u, v) \in E$, let $b(u, v)$ denote the balance of node u, and $c(u, v)$ denote the channel capacity. Therefore, we have $c(u, v) = b(u, v) + b(v, u)$. Recall that $c(u, v)$ is public information while $b(u, v)$ is private to nodes u and v.

The Max Flow Problem in LN. Given a graph G, a capacity function $c(.)$, a source $s \in V$, a destination $d \in V$, and a target channel $(a, b) \in E$, the problem is to find the maximum flow that s can send to d through the target channel (a, b). In addition to finding the maximum flow, the problem asks for a set of paths and payments that can achieve the max flow. Note that, in this problem, the balance function $b(.)$ is unknown. However, there is an access to an "oracle", which can be asked whether or not a given path can transfer a given payment. The oracle can also make a payment on a given path—assuming that the path can carry the payment—and accordingly update the balance of channels on the path. Note that this oracle is the Lighting Network itself.

5.2.2 Linear Programming

Given the balance function $b(.)$, the max flow problem can be solved in polynomial time using the following linear program (LP). This LP is similar to the one used to solve the traditional max flow problem with the main difference being that our LP maximizes the flow that enters the target channel while the traditional one maximizes the flow that leaves the source.

$$\begin{aligned} \max \quad & f_{ab} \\ \text{s.t.} \quad & 0 \leq f_{uv} \leq b(u, v), && \forall (u, v) \in E \\ & \sum_{u:(u,v)\in E} f_{uv} = \sum_{w:(v,w)\in E} f_{vw}, && \forall v \neq s, d \end{aligned} \tag{5.1}$$

In the above linear program, the variable f_{uv} represents the flow from u to v on channel $(u, v) \in E$. To avoid the target channel (a, b) to become the bottleneck, we set $b(a, b) = \infty$ in the above linear program.

LP Advantages. As mentioned earlier, in practice we do not have a direct access to the balance function $b(.)$ (we only have access to the oracle). Nevertheless, we can still benefit from solving the above LP by approximating the function $b(.)$. For instance, we may assume that $b(u, v) = 0.1 \times c(u, v)$ for every channel $(u, v) \neq (a, b)$. This assumption may not hold for all channels although the assumption is somewhat conservative—for each channel (u, v) either $b(u, v) \geq 0.5 \times c(u, v)$ or $b(v, u) \geq 0.5 \times c(u, v)$. Nevertheless, if by using the above assumption, we find that the max flow that can go through the target channel is considerably higher than, say, the capacity of the target channel, we can most likely discover the balance of the target channel in practice. The second advantage of this approach is that it can significantly speed up balance discovery because 1) we can use the method to pre-compute paths and payments, and 2) transmit all these payments in parallel. Note that if the max flow that Torrent can send through the target channel is higher than the balance of the channel, the balance of the channel can be safely assumed to be equal to the total amount of payments received at the receiver.

5.2.3 Greedy Algorithm

Our second algorithm is a greedy algorithm that works in iterations. In the first iteration, the algorithm finds a maximal set of disjoint paths[2] that go through the target channel. It then transmits payments on the selected paths until it saturates every single one of them. Before moving to the next iteration, the algorithm removes bottleneck channels, i.e. those from which it received an error message. For instance, if the algorithm receives an error message from node u indicating an insufficient balance on channel (u, v), then the algorithm removes the channel (u, v) in the direction of u to v. The algorithm, however, keeps the other direction (i.e. v to u) of channel (u, v) as the channel may be able to transfer payments in this direction.

After removing channels, the algorithm proceeds to the next iteration and repeats the above process. The algorithm terminates as soon as it concludes

[2]These paths share the target channel as well as the source and destination channels; otherwise, they are disjoint.

that it cannot transfer more payments though the target channel. This happens if it receives an error message indicating an insufficient balance on the target channel, or if the algorithm is unable to transmit more payments using new paths. When the algorithm terminates, it returns the amount of total payments collected at the receiver as the balance of the target channel.

Example 5 *Consider the simple payment network shown in Figure 5.1. If we run the greedy algorithm on this network, the algorithm can select paths*

$$p_1 = (n_1, n_2, Alice, Bob, n_4, n_6)$$

and

$$p_2 = (n_1, n_3, Alice, Bob, n_5, n_6)$$

in its first iteration. In this network, the algorithm can send up to 60k on path p_1 and up to 160k on path p_2. Since the sum of these two payments is higher than Alice's balance of 200k, at some point in this iteration the algorithm may receive an error message from Alice indicating insufficient balance. If this happens, the algorithm terminates and returns 200k as Alice's balance since this is the maximum it could transfer through her channel.

If Alice avoids sending error messages, the algorithm will move to next iterations and tries to find new paths and send new payments to increase the total received amount of 200k. Ultimately, the algorithm will terminate. Clearly, the algorithm can never deliver more than 200k to its receiver because Alice's channel is the bottleneck. Therefore, when the algorithm terminates, it will return the max amount it could deliver to its receiver (i.e. 200k) as Alice's balance.

5.3 Simulation Results

To evaluate the performance of Torrent, we downloaded a snapshot of the Lightning Network[3], which included 9169 nodes, and 76856 channels. We used the snapshot to generate the topology graph G and the capacity function $c(.)$.

[3]The snapshot was downloaded in March 2021.

The snapshot does not include the balance information as this information is private. Therefore, we used the following two methods to assign balances to channels:

- **Fractional Balances.** In this method, balances of every channel $(u, v) \in E$, except the target channel, are set to a fixed fraction of the channel capacity. That is, for every $(u, v) \in E$ both $b(u, v)$ and $b(v, u)$ are set to $\alpha \cdot c(u, v)$, where $\alpha < 0.5$ is a fixed number.
- **Random Balances.** This method selects balance of every channel (u, v), except the source and destination channels, uniformly at random from the interval $[0, c(u, v)]$.

To run Torrent Type A, we open two channels with the two nodes that have the highest degrees in the network. For Torrent Type B, we use only one of these two channels. In our simulations, we assume that the opened channels have enough balances.

5.3.1 Linear Programming

In our first set of simulations, we set the balances of all channels (except the source, target and destination channels) to a fraction $0.05 \leq \alpha \leq 0.3$ of their capacity. Then, we run Torrents Type A and B by solving the max flow problem using linear programming. We marked a target channel as *discoverable* if Torrent could generate a flow of at least equal to the capacity of the target channel. Note that this amount of flow is guaranteed to be at least equal to the balance of the target channel (no matter what the balance is), because balances of every channel are capped by the channel capacity.

Figure 5.2 shows the percentage of discoverable channels by Torrents Type A and B, respectively. For instance, if we set $\alpha = 0.05$—that is we set the two balances of every channel to 5% of the channel capacity—then Torrent Type A and B can, respectively, discover balances of 84% and 94.9% of all channels in the network. Note that with such low balances set for channels, the existing channel discovery methods are unlikely to discover the balance of any target channel unless they open a channel directly connected to the target channel.

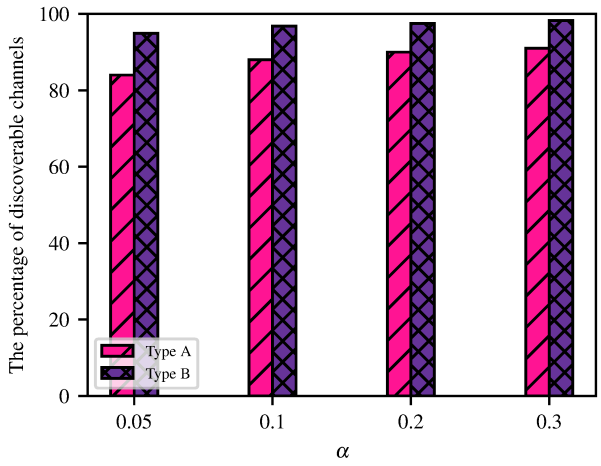


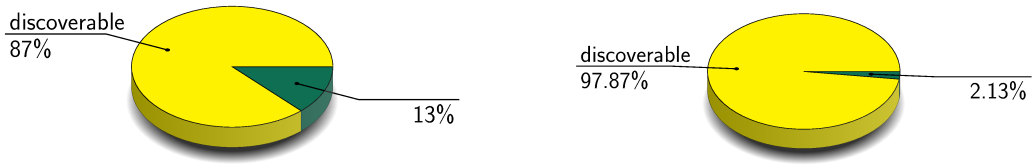
Figure 5.2: The percentage of discoverable channels by Torrents.

5.3.2 Greedy Algorithm

To evaluate the performance of Torrent powered by the greedy algorithm, we used the Random Balances. In this set of simulations, a target channel was marked as *discoverable* if Torrent could generate a flow that is at least equal to the balance of the target channel. Figure 5.3 shows the percentage of channels that could be discovered by Torrents Type A and B, respectively. This percentage is about 97.8% in the case of Torrent Type B. This is remarkable as, in Type B, Torrent only uses one channel, yet it can discover balances of nearly all the channels in the network.

As stated earlier, Torrent Type A is expected to discover a smaller percentage of balances that Type B. This is because Torrent Type A has to not only deliver a large flow of payments to the target channel, but also needs to deliver this large flow to its own receiver. To improve the performance of Torrent Type A, we can use multiple receivers (in our simulations, we used only a single receiver).

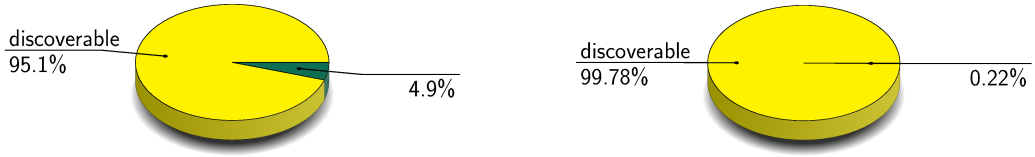
An interesting question here is how well the greedy algorithm performs compared to the optimal solution. To answer this question, we used a linear program, with the same balance function used in the greedy algorithm, to find the maximum number of discoverable channels. Recall that, given the balance function, linear programming can find the optimal solution to the



(a) Type A (b) Type B

Figure 5.3: The percentage of discoverable channels by Torrents Type A and B when they use the greedy algorithm.

max flow problem. The results of this simulation are shown in Figure 5.4. A parallel comparison between Figures 5.3 and 5.4 shows the greedy algorithm can discover balances of the vast majority of the channels that are possibly discoverable. This is particularly the case for Torrent Type B.



(a) Type A (b) Type B

Figure 5.4: The optimal percentage of discoverable channels.

5.4 Conclusion

We proposed Torrent, a powerful balance discovery approach. Unlike the existing balance discovery methods that use single-path payments, Torrent uses multi-path payments. This allows a single node to push a large flow of payments through any target channel. Torrent can work even when the Lightning Network does not transmit error messages to indicate insufficient balances. It is because Torrent can estimate a balance by calculating the maximum amount of fund that it can deliver to its receiver(s) through the target channel. For many channels in the Lightning Network, this maximum is equal to the balance of the target channel thanks to the large flow that Torrent can generate. Another advantage of Torrent is that, unlike the existing methods, it transmits payments in parallel, which can significantly speed up the balance discovery process.

Chapter 6

Conclusions & Future Works

Payment channel networks (PCNs) are among the most promising solutions to scale blockchains. These networks have a lot of room for improvement with regards to efficiency, privacy and security. In this thesis, we studied, analyzed and improved payment channels and PCNs by 1) designing a watchtower solution based on our novel reputation system (Hashcashed reputation); 2) increasing the payment success rate in PCNs through the design of a multi-path payment method with redundancy (Spear); and 3) designing a powerful method to discover balances of channels in PCNs (Torrent).

Hashcashed Reputation. We proposed a reputation system based on proof-of-work to encourage well-behaviour and inspire the spirit of competition in online marketplaces. To punish misbehaviour our proposed system maintains a proof of misbehaviour as a record in a distributed storage. This is an advantage compared to other reputation systems since our proposed system does not depend on a blockchain or smart contracts. In our system, it is enough to store each record in at least one node. This is an easy requirement to achieve since there is at least one party who is strongly motivated to store and (re)distribute it for each record. Finally, to showcase our reputation system, we designed two open markets of watchtowers and blockchain-based timestamping. We showed that our reputation system motivates watchtowers (or timestamp servers) not only to behave according to their obligation but also compete with each other by progressively improving their reputation and cutting their service fees.

Spear. In this work, we showed that redundant partial payments can remarkably improve the success probability of payments. We proposed Spear, a simple multi-path payment method with redundancy, and showed how it can be implemented in the Lightning Network. Moreover, we compared Spear with Boomerang, the only existing multi-path payment with redundancy in the literature. We showed that Spear outperforms Boomerang in terms of delay, ease of implementation, and computational requirement. The approach used in Spear can be extended to other payment methods, such as packet-switch routing methods, for further performance improvement.

Torrent. In the third work, we proposed Torrent, a novel and powerful balance discovery approach. Despite the existing balance discovery methods that use single-path payments, Torrent takes advantage of multi-path payments. This enables a single node to send a large flow of payments through any target channel. Torrent can function properly even when the Lightning Network does not convey error messages to reflect insufficient balances. It is because Torrent can estimate a balance by calculating the maximum amount of funds that it can deliver to its receiver(s) through the target channel. For many channels in the Lightning Network, this maximum is equal to the balance of the target channel thanks to the large flow that Torrent can generate. Despite the existing methods, Torrent transmits payments in parallel, which notably hasten the balance discovery process.

Future Works. Our proposed reputation system presented in Chapter 3 is flexible, and can be customized to achieve different needs. For instance, the system does not impose any specific selection method. One can, therefore, design a customized selection method to achieve a certain objective in the market.

Another possible future research is to relax the system reliance on the security parameter k. One approach is to ask clients to publish a digest of their ongoing committed contracts. This way, a new client can, for example, avoid a server whose total commitment value is greater than its reputation cost. Fortunately, clients have incentive to ask servers to sign such digests. They also have incentive to provide these digests to other servers. It is because

a client does not want its server to over-commit as it increases the risk of a successful bribery. Also, a server has an incentive to record the digests of other servers, as these digests can show a potential customer that other servers have already committed to many contracts (another means to attract a customer).

An interesting future work for Torrent (Chapter 5) is to find an efficient algorithm to solve the max flow problem in the Lightning Network. The algorithm can receive the network topology and channel capacities (but not balances) as input. It also has access to an oracle that answers to questions on whether or not a path p can transfer an amount m, and if so, can transfer the payment by updating balances of the channels on path p. Unlike the existing maximum flow solution, the efficiency of the algorithm will be determined by the number of times it accesses the oracle (rather than the algorithm's time complexity).

Bibliography

[1] S. Rahimpour and M. Khabbazian, "Hashcashed Reputation with Application in Designing Watchtowers," in *IEEE International Conference on Blockchain and Cryptocurrency, ICBC*, IEEE, 2021, pp. 1–9. DOI: 10.1109/ICBC51069.2021.9461123. [Online]. Available: https://doi.org/10.1109/ICBC51069.2021.9461123.

[2] S. Rahimpour and M. Khabbazian, "Spear: fast multi-path payment with redundancy," in *AFT '21: 3rd ACM Conference on Advances in Financial Technologies*, ACM, 2021, pp. 183–191. DOI: 10.1145/3479722.3480997. [Online]. Available: https://doi.org/10.1145/3479722.3480997.

[3] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A Secure Sharding Protocol For Open Blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2016, pp. 17–30. DOI: 10.1145/2976749.2978389. [Online]. Available: https://doi.org/10.1145/2976749.2978389.

[4] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding," in *IEEE Symposium on Security and Privacy, SP*, IEEE Computer Society, 2018, pp. 583–598. DOI: 10.1109/SP.2018.000-5. [Online]. Available: https://doi.org/10.1109/SP.2018.000-5.

[5] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timon, and P. Wuille, *Enabling blockchain innovations with pegged sidechains*, http://www.opensciencereview.com/papers/123/enablingblockchain-innovations-with-pegged-sidechains, 2014.

[6] A. Hertig, *Bitcoin Lightning Fraud? Laolu Is Building a 'Watchtower' to Fight It*, https://www.coindesk.com/laolu-building-watchtower-fight-bitcoin-lightning-fraud, Last Accessed: 2018-02-22.

[7] P. McCorry, S. Bakshi, I. Bentov, S. Meiklejohn, and A. Miller, "Pisa: Arbitration Outsourcing for State Channels," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT*, ACM, 2019, pp. 16–30. DOI: 10.1145/3318041.3355461. [Online]. Available: https://doi.org/10.1145/3318041.3355461.

[8] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, "Sprites and State Channels: Payment Networks that Go Faster Than Lightning," in *23rd International Conference on Financial Cryptography and Data Security, FC*, ser. Lecture Notes in Computer Science, vol. 11598, Springer, 2019, pp. 508–526. DOI: 10.1007/978-3-030-32101-7_30. [Online]. Available: https://doi.org/10.1007/978-3-030-32101-7%5C_30.

[9] G. Avarikioti, F. Laufenberg, J. Sliwinski, Y. Wang, and R. Wattenhofer, "Towards Secure and Efficient Payment Channels," *CoRR*, 2018. arXiv: 1811.12740. [Online]. Available: http://arxiv.org/abs/1811.12740.

[10] T. Dryja and S. B. Milano, "Unlinkable outsourced channel monitoring," *Talk transcript*, 2016. [Online]. Available: https://diyhpl.us/wiki/transcripts/scalingbitcoin/milan/unlinkable-outsourced-channel-monitoring/.

[11] V. K. Bagaria, J. Neu, and D. Tse, "Boomerang: Redundancy Improves Latency and Throughput in Payment-Channel Networks," in *24th International Conference on Financial Cryptography and Data Security, FC*, ser. Lecture Notes in Computer Science, vol. 12059, Springer, 2020, pp. 304–324. DOI: 10.1007/978-3-030-51280-4_17. [Online]. Available: https://doi.org/10.1007/978-3-030-51280-4%5C_17.

[12] J. Poon and T. Dryja, *The bitcoin lightning network: Scalable off-chain instant payments*, 2016.

[13] J. Herrera-Joancomartí, G. Navarro-Arribas, A. Ranchal-Pedrosa, C. Pérez-Solà, and J. García-Alfaro, "On the Difficulty of Hiding the Balance of Lightning Network Channels," in *Proceedings of the ACM Asia Conference on Computer and Communications Security, AsiaCCS*, ACM, 2019, pp. 602–612. DOI: 10.1145/3321705.3329812. [Online]. Available: https://doi.org/10.1145/3321705.3329812.

[14] S. Tikhomirov, R. Pickhardt, A. Biryukov, and M. Nowostawski, "Probing Channel Balances in the Lightning Network," *CoRR*, 2020. arXiv: 2004.00333. [Online]. Available: https://arxiv.org/abs/2004.00333.

[15] U. Nisslmueller, K.-T. Foerster, S. Schmid, and C. Decker, "Toward Active and Passive Confidentiality Attacks on Cryptocurrency Off-chain Networks," in *Proceedings of the 6th International Conference on Information Systems Security and Privacy, ICISSP*, SCITEPRESS, 2020, pp. 7–14. DOI: 10.5220/0009429200070014. [Online]. Available: https://doi.org/10.5220/0009429200070014.

[16] Y. Qiao, K. Wu, and M. Khabbazian, "Non-Intrusive and High-Efficient Balance Tomography in the Lightning Network," in *Proceedings of the ACM Asia Conference on Computer and Communications Security, AsiaCCS*, ACM, 2021, pp. 832–843. DOI: 10.1145/3433210.3453089. [Online]. Available: https://doi.org/10.1145/3433210.3453089.

[17] K. Fan, S. Wang, Y. Ren, H. Li, and Y. Yang, "MedBlock: Efficient and Secure Medical Data Sharing Via Blockchain," *Journal of medical systems*, vol. 42, no. 8, pp. 1–11, 2018. [Online]. Available: https://doi.org/10.1007/s10916-018-0993-7.

[18] U. W. Chohan, "Non-Fungible Tokens: Blockchains, Scarcity, and Value," *Critical Blockchain Research Initiative (CBRI) Working Papers*, 2021.

[19] Q. Wang, X. Zhu, Y. Ni, L. Gu, and H. Zhu, "Blockchain for the IoT and industrial IoT: A review," *Internet of Things*, vol. 10, p. 100 081, 2020. [Online]. Available: https://doi.org/10.1016/j.iot.2019.100081.

[20] F. S. Hardwick, A. Gioulis, R. N. Akram, and K. Markantonakis, "E-Voting With Blockchain: An E-Voting Protocol with Decentralisation and Voter Privacy," in *IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data*, IEEE, 2018, pp. 1561–1567. [Online]. Available: https://doi.org/10.1109/Cybermatics%5C_2018.2018.00262.

[21] I. Karamitsos, M. Papadaki, N. B. Al Barghuthi, *et al.*, "Design of the blockchain smart contract: A use case for real estate," *Journal of Information Security*, vol. 9, no. 03, p. 177, 2018.

[22] R. C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," in *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques*, ser. Lecture Notes in Computer Science, vol. 293, Springer, 1987, pp. 369–378. DOI: 10.1007/3-540-48184-2_32. [Online]. Available: https://doi.org/10.1007/3-540-48184-2%5C_32.

[23] Y. Kwon, H. Kim, J. Shin, and Y. Kim, "Bitcoin vs. Bitcoin Cash: Coexistence or Downfall of Bitcoin Cash?" In *IEEE Symposium on Security and Privacy, SP*, IEEE, 2019, pp. 935–951. DOI: 10.1109/SP.2019.00075. [Online]. Available: https://doi.org/10.1109/SP.2019.00075.

[24] Y. Sompolinsky and A. Zohar, "Secure High-Rate Transaction Processing in Bitcoin," in *19th International Conference on Financial Cryptography and Data Security, FC*, ser. Lecture Notes in Computer Science, vol. 8975, Springer, 2015, pp. 507–527. DOI: 10.1007/978-3-662-47854-7_32. [Online]. Available: https://doi.org/10.1007/978-3-662-47854-7%5C_32.

[25] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, "Perun: Virtual Payment Hubs over Cryptocurrencies," in *IEEE Symposium on Security and Privacy, SP*, IEEE, 2019, pp. 106–123. DOI: 10.1109/SP.2019.00020. [Online]. Available: https://doi.org/10.1109/SP.2019.00020.

[26] *Raiden Network*, https://raiden.network/, Last Accessed: 2020-08-19, 2018.

[27] J. Poon and V. Buterin, "Plasma: Scalable autonomous smart contracts," *White paper*, pp. 1–47, 2017.

[28] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "SoK: Layer-Two Blockchain Protocols," in *24th International Conference on Financial Cryptography and Data Security, FC*, J. Bonneau and N. Heninger, Eds., ser. Lecture Notes in Computer Science, vol. 12059, Springer, 2020, pp. 201–226. DOI: 10.1007/978-3-030-51280-4_12. [Online]. Available: https://doi.org/10.1007/978-3-030-51280-4%5C_12.

[29] C. Decker and R. Wattenhofer, "A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels," in *Stabilization, Safety, and Security of Distributed Systems - 17th International Symposium, SSS*, ser. Lecture Notes in Computer Science, vol. 9212, Springer, 2015, pp. 3–18. DOI: 10.1007/978-3-319-21741-3_1. [Online]. Available: https://doi.org/10.1007/978-3-319-21741-3%5C_1.

[30] A. Schrijver, "On the history of the transportation and maximum flow problems," *Mathematical programming*, vol. 91, no. 3, pp. 437–445, 2002. DOI: 10.1007/s101070100259. [Online]. Available: https://doi.org/10.1007/s101070100259.

[31] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian journal of Mathematics*, vol. 8, pp. 399–404, 1956.

[32] A. Paithankar and S. Chatterjee, "Open pit mine production schedule optimization using a hybrid of maximum-flow and genetic algorithms," *Applied Soft Computing*, vol. 81, 2019. DOI: 10.1016/j.asoc.2019.105507. [Online]. Available: https://doi.org/10.1016/j.asoc.2019.105507.

[33] Q. Zhu, M. Chen, B. Feng, Y. Zhou, M. Li, Z. Xu, Y. Ding, M. Liu, W. Wang, and X. Xie, "Optimized Spatiotemporal Data Scheduling Based on Maximum Flow for Multilevel Visualization Tasks," *ISPRS International Journal of Geo-Information*, vol. 9, no. 9, p. 518, 2020. DOI: 10.3390/ijgi9090518. [Online]. Available: https://doi.org/10.3390/ijgi9090518.

[34] J. I. Orlicki, "Generalized Minimum Cost Flow and Arbitrage in Bitcoin Debit and Custodian Networks," in *I Simposio Argentino de Informática Industrial e Investigación Operativa (SIIIO 2018)-JAIIO 47 (CABA)*, 2018.

[35] A. Armbruster, M. R. Gosnell, B. M. McMillin, and M. L. Crow, "Power Transmission Control Using Distributed Max Flow," in *29th Annual International Computer Software and Applications Conference, COMPSAC*, IEEE Computer Society, 2005, pp. 256–263. [Online]. Available: https://doi.org/10.1109/COMPSAC.2005.121.

[36] C. Huang, Z. Wang, H. Chen, Q. Hu, Q. Zhang, W. Wang, and X. Guan, "RepChain: A Reputation-Based Secure, Fast, and High Incentive Blockchain System via Sharding," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4291–4304, 2021. DOI: 10.1109/JIOT.2020.3028449. [Online]. Available: https://doi.org/10.1109/JIOT.2020.3028449.

[37] M. Li, J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, J.-N. Liu, Y. Xiang, and R. H. Deng, "CrowdBC: A Blockchain-Based Decentralized Framework for Crowdsourcing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 6, pp. 1251–1266, 2019. DOI: 10.1109/TPDS.2018.2881735. [Online]. Available: https://doi.org/10.1109/TPDS.2018.2881735.

[38] M. Nojoumian, A. Golchubian, L. Njilla, K. Kwiat, and C. Kamhoua, "Incentivizing blockchain miners to avoid dishonest mining strategies by a reputation-based paradigm," in *Science and Information Conference*, Springer, 2018, pp. 1118–1134.

[39] M. Orlovsky and A. Sobol, *ERC-1329: Inalienable Reputation Token*, https://github.com/ethereum/EIPs/issues/1329, Accessed: 2020-08-19.

[40] T. Salman, R. Jain, and L. Gupta, "A Reputation Management Framework for Knowledge-Based and Probabilistic Blockchains," in *IEEE International Conference on Blockchain, Blockchain*, IEEE, 2019, pp. 520–527. DOI: 10.1109/Blockchain.2019.00078. [Online]. Available: https://doi.org/10.1109/Blockchain.2019.00078.

[41] Q. Zhuang, Y. Liu, L. Chen, and Z. Ai, "Proof of Reputation: A Reputation based Consensus Protocol for Blockchain Based Systems," in *Proceedings of the International Electronics Communication Conference*, 2019, pp. 131–138.

[42] J. Yu, D. Kozhaya, J. Decouchant, and P. J. E. Verissimo, "RepuCoin: Your Reputation Is Your Power," *IEEE Transactions on Computers*, vol. 68, no. 8, pp. 1225–1237, 2019. DOI: 10.1109/TC.2019.2900648. [Online]. Available: https://doi.org/10.1109/TC.2019.2900648.

[43] R. Dennis and G. Owen, "Rep on the block: A next generation reputation system based on the blockchain," in *10th International Conference for Internet Technology and Secured Transactions, ICITST*, IEEE, 2015, pp. 131–138. DOI: 10.1109/ICITST.2015.7412073. [Online]. Available: https://doi.org/10.1109/ICITST.2015.7412073.

[44] A. Schaub, R. Bazin, O. Hasan, and L. Brunie, "A Trustless Privacy-Preserving Reputation System," in *ICT Systems Security and Privacy Protection - 31st IFIP TC 11 International Conference, SEC*, J.-H. Hoepman and S. Katzenbeisser, Eds., ser. IFIP Advances in Information and Communication Technology, vol. 471, Springer, 2016, pp. 398–411. DOI: 10.1007/978-3-319-33630-5_27. [Online]. Available: https://doi.org/10.1007/978-3-319-33630-5%5C_27.

[45] M. Sharples and J. Domingue, "The Blockchain and Kudos: A Distributed System for Educational Record, Reputation and Reward," in *11th European Conference on Technology Enhanced Learning, EC-TEL*, K. Verbert, M. Sharples, and T. Klobucar, Eds., ser. Lecture Notes in Computer Science, vol. 9891, Springer, 2016, pp. 490–496. DOI: 10.1007/978-3-319-45153-4_48. [Online]. Available: https://doi.org/10.1007/978-3-319-45153-4%5C_48.

[46] C. Dwork and M. Naor, "Pricing via Processing or Combatting Junk Mail," in *Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference*, ser. Lecture Notes in Computer Science, vol. 740, Springer, 1992, pp. 139–147. DOI: 10.1007/3-540-48071-4_10. [Online]. Available: https://doi.org/10.1007/3-540-48071-4%5C_10.

[47] A. Back, "Hash cash: A partial hash collision based postage scheme," *URL http://www. hashcash. org*, 2001.

[48] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, pp. 1–9, 2008.

[49] D. Mankins, R. Krishnan, C. Boyd, J. Zao, and M. Frentz, "Mitigating Distributed Denial of Service Attacks with Dynamic Resource Pricing," in *17th Annual Computer Security Applications Conference (ACSAC)*, IEEE Computer Society, 2001, pp. 411–421. DOI: 10.1109/ACSAC.2001.991558. [Online]. Available: https://doi.org/10.1109/ACSAC.2001.991558.

[50] M. Baza, M. Nabil, M. M. E. A. Mahmoud, N. Bewermeier, K. Fidan, W. Alasmary, and M. Abdallah, "Detecting sybil attacks using proofs of work and location in vanets," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2020. DOI: 10.1109/TDSC.2020.2993769.

[51] N. Borisov, "Computational Puzzles as Sybil Defenses," in *Sixth IEEE International Conference on Peer-to-Peer Computing (P2P)*, IEEE Computer Society, 2006, pp. 171–176. DOI: 10.1109/P2P.2006.10. [Online]. Available: https://doi.org/10.1109/P2P.2006.10.

[52] H. Rowaihy, W. Enck, P. D. McDaniel, and T. L. Porta, "Limiting Sybil Attacks in Structured P2P Networks," in *26th IEEE International Conference on Computer Communications (INFOCOM), Joint Conference of the IEEE Computer and Communications Societies,*, IEEE, 2007, pp. 2596–2600. DOI: 10.1109/INFCOM.2007.328. [Online]. Available: https://doi.org/10.1109/INFCOM.2007.328.

[53] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer, "Karma: A secure economic framework for peer-to-peer resource sharing," in *Workshop on Economics of Peer-to-peer Systems*, vol. 35, 2003, pp. 1–6.

[54] A. Biryukov and I. Pustogarov, "Proof-of-Work as Anonymous Micropayment: Rewarding a Tor Relay," in *19th International Conference on Financial Cryptography and Data Security FC*, ser. Lecture Notes in Computer Science, vol. 8975, Springer, 2015, pp. 445–455. DOI: 10.1007/978-3-662-47854-7_27. [Online]. Available: https://doi.org/10.1007/978-3-662-47854-7%5C_27.

[55] M. Hearn, "Anti DoS for tx replacement," *bitcoin-dev mailing list*, 2013.

[56] J. Spilman, "Anti DoS for tx replacement," *bitcoin-dev mailing list*, 2013.

[57] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," in *Ethereum project yellow paper*, vol. 151, 2014, pp. 1–32.

[58] *Blockstream*, https://github.com/ElementsProject/lightning/, Last Accessed: 2020-08-19.

[59] Eclair, https://github.com/ACINQ/eclair, Last Accessed: 2020-08-19.

[60] *Lightning Labs*, https://github.com/lightningnetwork/lnd, Last Accessed: 2020-08-19.

[61] Z. Avarikioti, E. Kokoris-Kogias, R. Wattenhofer, and D. Zindros, "Brick: Asynchronous Incentive-Compatible Payment Channels," in *25th International Conference on Financial Cryptography and Data Security, FC*, ser. Lecture Notes in Computer Science, vol. 12675, Springer, 2021, pp. 209–230. DOI: 10.1007/978-3-662-64331-0_11. [Online]. Available: https://doi.org/10.1007/978-3-662-64331-0%5C_11.

[62] M. Khabbazian, T. Nadahalli, and R. Wattenhofer, "Outpost: A Responsive Lightweight Watchtower," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT*, ACM, 2019, pp. 31–40. DOI: 10.1145/3318041.3355464. [Online]. Available: https://doi.org/10.1145/3318041.3355464.

[63] Z. Avarikioti, O. S. T. Litos, and R. Wattenhofer, "Cerberus Channels: Incentivizing Watchtowers for Bitcoin," in *24th International Conference on Financial Cryptography and Data Security, FC*, ser. Lecture Notes in Computer Science, vol. 12059, Springer, 2020, pp. 346–366. DOI: 10.1007/978-3-030-51280-4_19. [Online]. Available: https://doi.org/10.1007/978-3-030-51280-4%5C_19.

[64] MPP, *routing: multi-part mpp*, https://github.com/lightningnetwork/lnd/pull/3967, Last Accessed: 2020-11-12, 2020.

[65] G. Malavolta, P. Moreno-Sanchez, A. Kate, and M. Maffei, "SilentWhispers: Enforcing Security and Privacy in Decentralized Credit Networks," in *Proceedings of the 24th Annual Network and Distributed System Security Symposium, NDSS*, The Internet Society, 2017, pp. 1–15. [Online]. Available: https://www.ndss-symposium.org/ndss2017/ndss-2017-

programme/silentwhispers-enforcing-security-and-privacy-decentralized-credit-networks/.

[66] S. Roos, P. Moreno-Sanchez, A. Kate, and I. Goldberg, "Settling Payments Fast and Private: Efficient Decentralized Routing for Path-Based Transactions," in *Proceedings of the 25th Annual Network and Distributed System Security Symposium, NDSS*, The Internet Society, 2018, pp. 1–15. [Online]. Available: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018%5C_09-3%5C_Roos%5C_paper.pdf.

[67] V. Sivaraman, S. B. Venkatakrishnan, K. Ruan, P. Negi, L. Yang, R. Mittal, G. C. Fanti, and M. Alizadeh, "High Throughput Cryptocurrency Routing in Payment Channel Networks," in *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI*, USENIX Association, 2020, pp. 777–796. [Online]. Available: https://www.usenix.org/conference/nsdi20/presentation/sivaraman.

[68] V. Sivaraman, S. B. Venkatakrishnan, M. Alizadeh, G. C. Fanti, and P. Viswanath, "Routing Cryptocurrency with the Spider Network," in *Proceedings of the 17th ACM Workshop on Hot Topics in Networks, HotNets*, ACM, 2018, pp. 29–35. DOI: 10.1145/3286062.3286067. [Online]. Available: https://doi.org/10.1145/3286062.3286067.

[69] D. Piatkivskyi and M. Nowostawski, "Split payments in payment networks," in *Proceedings of the International Workshops on Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS*, ser. Lecture Notes in Computer Science, vol. 11025, Springer, 2018, pp. 67–75. DOI: 10.1007/978-3-030-00305-0_5. [Online]. Available: https://doi.org/10.1007/978-3-030-00305-0%5C_5.

[70] O. Osuntokun, *AMP: Atomic Multi-Path Payments over Lightning*, https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html, 2018.

[71] P. Hoenisch and I. Weber, "AODV-Based Routing for Payment Channel Networks," in *Proceedings of the First International Conference on BlockchainICBC, Held as Part of the Services Conference Federation, SCF*, ser. Lecture Notes in Computer Science, vol. 10974, Springer, 2018, pp. 107–124. DOI: 10.1007/978-3-319-94478-4_8. [Online]. Available: https://doi.org/10.1007/978-3-319-94478-4%5C_8.

[72] L. Eckey, S. Faust, K. Hostáková, and S. Roos, "Splitting Payments Locally While Routing Interdimensionally," *IACR Cryptol. ePrint Arch.*, p. 555, 2020. [Online]. Available: https://eprint.iacr.org/2020/555.

[73] P. Prihodko, S. Zhigulin, M. Sahno, A. Ostrovskiy, and O. Osuntokun, "Flare: An approach to routing in lightning network," *White Paper*, pp. 1–40, 2016.

[74] B. Viswanath, M. Mondal, P. K. Gummadi, A. Mislove, and A. Post, "Canal: scaling social network-based Sybil tolerance schemes," in *Proceedings of the 7th European Conference on Computer Systems*, Bern, Switzerland: ACM, 2012, pp. 309–322. DOI: 10.1145/2168836.2168867. [Online]. Available: https://doi.org/10.1145/2168836.2168867.

[75] P. Moreno-Sanchez, A. Kate, M. Maffei, and K. Pecina, "Privacy Preserving Payments in Credit Networks: Enabling trust with privacy in online marketplaces," in *Proceedings of the 22nd Annual Network and Distributed System Security Symposium, NDSS*, The Internet Society, 2015, pp. 1–15. [Online]. Available: https://www.ndss-symposium.org/ndss2015/privacy-preserving-payments-credit-networks-enabling-trust-privacy-online-marketplaces.

[76] S. Dziembowski and P. Kedzior, "Ethna: Channel Network with Dynamic Internal Payment Splitting," *IACR Cryptol. ePrint Arch.*, pp. 1–18, 2020. [Online]. Available: https://eprint.iacr.org/2020/166.

[77] G. van Dam, R. A. Kadir, P. N. E. Nohuddin, and H. B. Zaman, "Improvements of the Balance Discovery Attack on Lightning Network Payment Channels," in *ICT Systems Security and Privacy Protection - 35th IFIP TC 11 International Conference, SEC*, ser. IFIP Advances in Information and Communication Technology, vol. 580, Springer, 2020, pp. 313–323. DOI: 10.1007/978-3-030-58201-2_21. [Online]. Available: https://doi.org/10.1007/978-3-030-58201-2%5C_21.

[78] G. Kappos, H. Yousaf, A. M. Piotrowska, S. Kanjalkar, S. Delgado-Segura, A. Miller, and S. Meiklejohn, "An Empirical Analysis of Privacy in the Lightning Network," in *25th International Conference on Financial Cryptography and Data Security, FC*, ser. Lecture Notes in Computer Science, vol. 12674, Springer, 2021, pp. 167–186. DOI: 10.1007/978-3-662-64322-8_8. [Online]. Available: https://doi.org/10.1007/978-3-662-64322-8%5C_8.

[79] A. Biryukov, G. Naumenko, and S. Tikhomirov, "Analysis and Probing of Parallel Channels in the Lightning Network," *IACR Cryptol. ePrint Arch.*, pp. 1–24, 2021. [Online]. Available: https://eprint.iacr.org/2021/384.

[80] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse Attacks on Bitcoin's Peer-to-Peer Network," in *24th USENIX Security Symposium, USENIX Security 15*, USENIX Association, 2015, pp. 129–144. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman.

[81] *BOLT 2: Peer Protocol for Channel Management*, https://ln.dev/read/02-peer-protocol, Last Accessed: 2020-10-26.

[82] A. Mizrahi and A. Zohar, "Congestion Attacks in Payment Channel Networks," in *25th International Conference on Financial Cryptography and Data Security, FC*, ser. Lecture Notes in Computer Science, vol. 12675, Springer, 2021, pp. 170–188. DOI: 10.1007/978-3-662-64331-0_9. [Online]. Available: https://doi.org/10.1007/978-3-662-64331-0%5C_9.

[83] A. C.-C. Yao, "On the Evaluation of Powers," *SIAM Journal on computing*, vol. 5, no. 1, pp. 100–103, 1976. DOI: 10.1137/0205008. [Online]. Available: https://doi.org/10.1137/0205008.

[84] Lightning-Network-Daemon, *lnd v0.13.0-beta*, https://github.com/lightningnetwork/lnd/releases/tag/v0.13.0-beta, 2021.