LightSync: Ultra Light Client for PoW Blockchains

by

Niusha Moshrefi

B.Sc., Sharif University of Technology, 2019

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE COLLEGE OF GRADUATE STUDIES
(Electrical Engineering)

The University of British Columbia
(Okanagan)

November 2022

The following individuals certify that they have read, and recommend to the College of Graduate Studies for acceptance, a thesis/dissertation entitled:

LightSync: Ultra Light Client for PoW Blockchains

submitted by **Niusha Moshrefi** in partial fulfillment of the requirements for the degree of **MASTER OF APPLIED SCIENCE**.

Examining Committee:

Chen Feng, School of Engineering
Supervisor

Anas Chaaban, School of Engineering
Supervisory Committee Member

Ahmad Al-Dabbagh, School of Engineering
Supervisory Committee Member

Abstract

Full nodes in a blockchain network store and verify a copy of the whole blockchain. Unlike full nodes, light clients are low-capacity devices that want to validate certain data on a blockchain. They query the data they want from a full node. If light clients do not verify the data they receive, full nodes might deceive them. SPV, introduced in the Bitcoin white paper, is a practical solution to this problem currently used in many PoW blockchains. In SPV, the resources needed to verify a full node's response grow linearly with the blockchain size, making it inefficient over the long run. Another issue with SPV is that the full nodes do not get compensated for the services they provide.

In this work, we introduce LIGHTSYNC, a *simple* and *cost-effective* solution for light clients to verify the inclusion of certain data in a PoW blockchain. LIGHTSYNC enjoys a number of salient features. First, the resources needed for running LIGHTSYNC remain *constant* no matter what the size of the blockchain is. Second, LIGHTSYNC is provably secure under the variable difficulty settings. Third, LIGHTSYNC can be implemented without the need for any excess built-in structure.

Lay Summary

In a blockchain network, everyone needs access to the honest data of the blockchain. Light nodes in a blockchain are nodes that have low storage and computation power. They cannot become a full node to store and verify the whole data of the blockchain because of their limited resources. They must query full nodes to get the data they want. However, full nodes might provide the light node with malicious data, therefore, light nodes should be able to verify the data they receive. In this work, we propose a simple and cost-efficient protocol to enable Proof-of-Work light nodes to access the data of the blockchain and make sure that this data is not corrupted by a malicious full node. Our solution has a constant cost regardless of the size of the blockchain.

Preface

All of the work presented in this thesis was conducted under the supervision of Prof. Chen Feng. The dissertation is based on the research done on interoperability of blockchains, light clients. The protocol design is done by M. Daneshpajooh and me. The Analysis, Implementation, Evaluation, and Discussion (from Chapter 5 to the end) is mostly done and written by me. Chen Feng was involved in revising the results, analysis, and the material. He also mentioned the ”chain-sewing attack” for FlyClient discussed in Chapter 2.

Table of Contents

List of Tables

List of Figures

Glossary

SPV Simple Payment Verification

POW Proof of Work

MMR Merkle Mountain Range

NIPOPOW Non-Interactive Proof of Proof of Work

Acknowledgement

I want to thank my supervisor, Chen Feng, for his guidance and mentorship during my studies. He was kind, considerate, and supportive. He let me take the process at my own pace. I also want to thank Mahyar Daneshpajooh whom I learned from a lot in this journey.

To my mother
Who always believed in me
And to Mahyar
Who was always there for me

Chapter 1

Introduction

1.1 Blockchains

Blockchain is a digital ledger that is maintained by a network of nodes. These nodes reach a consensus on the latest state of the ledger. The consensus mechanism can differ in different blockchains. There are several consensus mechanisms like Proof of Work, Proof of Stake, Proof of Storage, etc.

The digital ledger is a database that consists of blocks. Each block points to its previous block so that all the blocks form a single chain. Blocks contain transactions that change the state of the blockchain. Transactions are created and signed by users, then, they get propagated in the network. Miners are nodes that gather the transactions, seal them into a block and add that block to the blockchain.

Here, we focus on Proof of Work (POW) blockchains. In PoW blockchains, miners must solve a puzzle[1] to be able to seal the block they create, unless, other nodes in the network will not accept that block. This is to make sure that the miner has done enough work on the block. For each block, the first miner who solves the puzzle wins the reward of that block. Every node in the system accepts valid blocks that miners create to grow their local chain and stay up-to-date.

Solving the puzzle requires computation power. The more computation power

[1] Solving this puzzle means to find a proper nonce for which hash of the block header would be less than its target difficulty. Where block header consists of nonce and some other information. We will elaborate on this in Chapter 3.

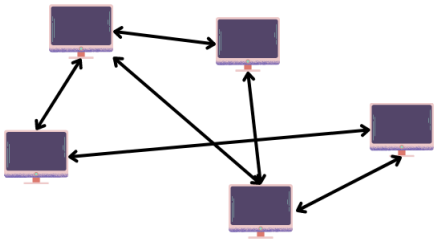

Figure 1.1: A peer-to-peer network of nodes maintaining the blockchain data

a node owns, the more blocks it can create and the more reward it can collect. Honest miners use their computation power to create valid blocks and earn rewards. However, there might be some malicious nodes creating invalid blocks that contain malicious transactions with contradicting data to the history of the blockchain. However, the system stays safe (i.e. malicious blocks get rejected by other nodes) as long as the total honest computing power is more than %50 of the total computing power.[2]

Bitcoin[17], is the first blockchain in the world introduced by Satoshi Nakamoto in 2007. It is a peer-to-peer payment system that aims to eliminate the need for a centralized banking system. Bitcoin uses PoW for its consensus mechanism.

1.2 Light Client Problem

In a blockchain network, we have different participants such as full nodes, miners, and light nodes. We have talked about miners. Full nodes are nodes that maintain a copy of the whole blockchain and relay data in the network. They need to have high storage, bandwidth, and computation power[3] since they need to store all the blocks and verify newly mined blocks to add them to their local chain. For example, as

[2]We emphasize that here the measure is on computing power rather than the number of the nodes. This gives the advantage of not being vulnerable against Sybil attacks.

[3]This means at least a few hundreds of gigabytes of storage, and the ability to compute millions of hashes for its verification.

of this writing, Ethereum's blockchain[9] size is more than 900 gigabytes [3]. Full nodes validate each and every block and the consensus mechanism.

However, some clients have access to limited resources[4]. These clients might still need instant access to certain data on a blockchain. Nowadays, there exist many such clients such as smartphones, IoT devices, and wearable devices who want to get data from a blockchain without storing and verifying the whole blockchain. They might need the data to calculate the balance of a user, make sure a specific transaction has been finalized on the blockchain, etc. These low-capacity clients are called *light clients*. In case a light client wants to verify data from multiple chains, the need for a cost-effective solution becomes more crucial.

Another concern in today's blockchain ecosystem is the interoperability of different blockchains [7, 10, 21]. Recently, the topic of cross-chain applications has attracted wide attention [2, 6, 14]. In order to enable blockchains to embed a client of another blockchain to facilitate cross-chain transactions and cross-chain data transfer, there is a need for an efficient solution for light clients.

One solution for the light clients who need access to the blockchain data is to send their queries to a trusted third party (who maintains the blockchain) and get the required data. However, this approach contradicts the very concept of decentralization in blockchains. Thus, a light client needs an efficient solution to obtain information from any full node and some proof to confirm its validity, inclusion in the blockchain, and finalization.

To better formulate the problem, we assume there exists some full nodes, and the light client wants to prove the inclusion of a predicate in the blockchain by interacting with these full clients. The light client cannot trust a full node to make sure which one is providing honest data.

1.3 Related Work

Nakamoto in the original Bitcoin whitepaper proposed a more efficient way for a light client to verify the inclusion of some data in the blockchain. Light clients can only store and verify the block headers of a blockchain. Ethereum block size is 508 B, so, currently, for an Ethereum light client to store all the block headers using this

[4]They might have limited storage, computation power, or bandwidth.

method, a storage size of nearly 7 gigabytes is needed. While this method reduces the storage needed for the client dramatically, it is still not a scalable solution. In fact, when the blockchain grows, the needed storage for the client grows linearly with it.

Recently, two sub-linear solutions have been proposed to solve this problem. NIPoPoWs and FlyClient introduce *superlight* blockchain clients that have a poly-logarithmic cost with respect to the size of the blockchain. However, they both need additional data structures to be included in the blocks of a PoW blockchain to work. Moreover, NIPoPoWs only works properly when the target difficulty of the blockchain remains constant. FlyClient uses another approach that takes variable target difficulties into account. Later, it was discovered that FlyClient is exposed to the chain-sewing attack. So, the need for a proper solution for superlight clients still exists.

Two important problems need to be tackled here. First, the cost of the solution should be minimized in terms of the required storage and the computation power for the light client. Second, an incentive mechanism should be designed for the full nodes to provide the data and the associated proof. This especially becomes more important when a full node confronts massive inquiries from light clients.

1.4 Our Contribution

In this thesis we propose LIGHTSYNC, a new protocol for light clients in PoW blockchains. The storage and computation power needed for the users in LIGHTSYNC do not grow with the length of the blockchain and remain constant. Thus, LIGHTSYNC dramatically reduces the resources that light clients need for confirming a predicate from a PoW blockchain. Especially for the light clients who want to confirm data from a large or a fast-growing blockchain, LIGHTSYNC is a promising solution to reduce the cost. For example, as of this writing, the length of the Ethereum blockchain is roughly 15 Million blocks. Assume the adversary mining power to the honest mining power to be $\frac{1}{2}$ and the adversary miners to use the same target difficulty as honest miners. In case the blockchain includes MMR root in the block header structure, to achieve the failure probability less than 2^{-50}, the expected number of block headers for LIGHTSYNC's proof will be 405. In Ta-

Table 1.1: Proof size (in MB) of SPV and LightSync for Ethereum blockchain

Blockchain length	**1 M**	**10 M**	**100 M**
SPV	508	5,080	50,800
LightSync	0.204	0.204	0.204

ble 1.1, we demonstrate the improvement in proof size in comparison to SPV. Our simple solution is based on realistic assumptions, making it easily applicable to current PoW blockchains. More importantly, we have analyzed the security of the protocol thoroughly [19] and considered all possible deployment issues. To sum up, LIGHTSYNC is the first protocol for light clients that has the minimum cost for its users and is provably secure.

In this chapter, we defined the problem, discussed some solutions, and introduced our contribution to the problem. In the next chapter, we will go through the basics of blockchains and some previous work to build a basis for protocol design and analysis. In Chapter 3 we introduce our model and define the problem. In Chapter 4, we propose the LIGHTSYNC protocol. Chapter 5 performs a thorough analysis of the protocol. Chapter 6 evaluates the proposed solution in practice, and at last, in Chapter 7, we discuss some applications and further improvements to the protocol. Chapter 8 concludes it all.

Chapter 2

Background

2.1 The Principles of a Blockchain

In this chapter, we aim to describe the fundamentals and previous works that will be a basis for the rest of the chapters.

2.1.1 PoW Blockchains

Let's first describe what a blockchain is and how it all started. Blockchain is a distributed ledger that is maintained by a peer-to-peer network. Nodes of the network reach consensus on the latest state of this ledger. The ledger consists of blocks, which are data structures containing transactions that change the state of the ledger.

The first blockchain in history is Bitcoin introduced by Satoshi Nakamoto in 2007. The goal of this system was to provide a peer-to-peer payment system that does not need a centralized authority to control it. In other words, Bitcoin eliminated the need for a trusted third party to solve the double-spending problem.

Bitcoin consensus mechanism is POW in which a node provides cryptographic proof that shows it has expended a specific computational effort in order to create a new block. This proof is easily verifiable by others and is provided with each newly created block to update the state of the blockchain. In this system, the nodes that create new blocks are called *miners*. It takes some time for a miner to do the necessary work, create the proof, and release a new block. The difficulty of

creating such proof is variable in Bitcoin and depends on the total computing power of the miners in the network. The difficulty adjusts so that on average the rate of newly created blocks remains constant. This adjustment is called the *re-targeting algorithm* of Bitcoin.

2.1.2 Block Structure

A blockchain consists of a chain of blocks, where blocks are data structures that contain transactions. Transactions change the state of a blockchain. On Bitcoin every user can have a pair of private and public keys, to be able to send and receive transactions. The public key is the address of the user on a blockchain, and the private key enables the user to sign transactions. Others can validate the signature and make sure they belong to the same pair of private and public keys.

A block mainly consists of two parts: block header, and transactions. Block header is what can be described as the metadata of the block. It contains the block version, the previous block header hash, Merkle root of the transactions, difficulty target, timestamp, and nonce. Nonce gets determined when the puzzle of the Proof of Work gets solved. This puzzle satisfies the condition that the hash of the block header is less than a specific threshold called target difficulty. The less the target difficulty is, the harder creating a new block will be. Merkle root of the transactions is a hash that acts as a commitment for the whole transactions that exist in that block. Using the Merkle tree built based on the transactions, one can create Merkle proofs that show the inclusion of a specific transaction in that block. Verifying a Merkle proof is a straightforward and low computation process.

2.1.3 Full Nodes And Light Clients

There are some nodes in the blockchain network who maintain the whole blockchain history and validate every new block to comply with it, then add that block to their storage. These nodes are called *full nodes*. Full nodes also relay data (i.e. blocks and transactions) in the peer-to-peer network. They need a huge storage and computation power for what they do.

If anyone wants to get access to blockchain data, they have to connect to the network of full nodes and query them for the data they want. However, they have to

verify the data they are receiving to make sure the provided data is not malicious. But, how can a node verify that? A naive approach would be to get the whole data of the blockchain from a full node and to validate each and every block from the bottom (genesis block) to the top (latest block of the blockchain at that time). This approach requires a lot of resources, as it requires the node to validate the Proof of Work for every block as well as the other elements of the block header. If such nodes want to become full nodes themselves, they inevitably have to do these heavy computations and dedicate enough storage. But, let us consider the case in which the node only wants to verify the inclusion of one transaction, or make sure if some block has been finalized. In this case, the node might not be capable of doing huge computations or dedicating lots of storage for the matter. We call these nodes with limited resources *light nodes*. Light nodes should be able to efficiently access the data they want from the blockchains.

In the section 2.2 we explain more about this issue and some better alternatives to our naive solution for it.

2.1.4 Forks

Blockchain forks are essentially a split in the blockchain network. It happens when the community makes a change to the basic rules of the blockchain and its protocol. Some portion of nodes update to the new rules and some remain the same as before. We have mainly two types of forks in blockchains: *hard forks*, and *soft forks*.

In hard forks, the new updates break the backward compatibility. However, in soft forks changes are backward-compatible and old nodes will still recognize the updated blocks as valid ones. On the other hand, the updated nodes see the old miners' blocks as invalid blocks.

In NIPoPoWs [22], Aggelos Kiayias et al. have introduced a new type of fork: *velvet forks*. In the next section, we will talk about velvet forks and how they work in more detail.

Velvet Forks

When a new update to the consensus layer is provided, the majority of miners are required to agree on it; even in the case of soft forks. Velvet forks, however, do

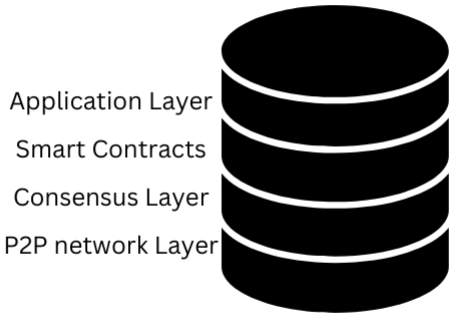

Figure 2.1: Different layers of a blockchain system including the consensus layer.

not modify the consensus layer of the network. They provide some modifications in the block data and only mandate the upgraded miners to include them in their blocks. While the rest of the network does not include the additional data into the blocks and does not verify its inclusion, treating them merely as comments. The changes here are backward-compatible, and both groups of miners (i.e. upgraded and non-upgraded) accept blocks mined by the other group. Therefore, the set of accepted blocks is not changed. This prevents chain splits and also allows the protocol to upgrade even if only a minority of miners upgrade.

Moreover, we have to be careful, since even if the additional data in the upgrade is invalid or malicious, upgraded nodes are forced to accept the blocks. Because validating the extra data is not added to the rule of consensus for miners.

2.2 Simple Payment Verification

The original Bitcoin white paper [17] has introduced *Simple Payment Verification (*SPV*) clients*, who are light clients performing a payment verification by storing and verifying all the block headers of a blockchain. Blocks of Bitcoin consist of two main parts: block header and transactions. The block header consists of

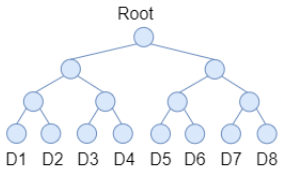

Figure 2.2: A Merkle tree constructed on top of the dataset {D1, D2, ..., D8}

some information about the block, in addition to a hash commitment of the whole transactions included in that block. The inclusion of a transaction in a block can efficiently be verified against the hash commitment provided in the block header. This hash commitment is the Merkle root that we discussed in the previous section.

Instead of storing and verifying the whole blockchain, SPV clients only verify and store the block headers of a blockchain. In this case, the SPV light client can still verify the inclusion of any part of the data in the blockchain using Merkle proofs; including transactions. This can be done by providing proof of inclusion for a transaction in the Merkle root included in the block header.

Let us take a closer look at how Merkle trees work to better understand this mechanism.

A Merkle tree is a hash-based binary-tree data structure. Suppose each leaf node is a hash of a transaction, each non-leaf node is a hash of its children, and each parent has a maximum number of 2 children. Each pair of nodes are recursively hashed until we reach the root node. The root hash is included in the block header of the block containing all those transactions. In Figure 2.2 you can see the structure of the tree and how the root gets formed.

To create a proof for some data included in the tree, it suffices to provide the hash of the data, as well as its path to the root. This way everyone who has the proof can re-produce the whole path and verify that the data was actually included in the Merkle tree with that specific Merkle root. The proof size is logarithmic with respect to the data size, which makes proof verification an easy process.

To sum up, the SPV approach is cheaper than storing all the blocks of the blockchain and acting as a full node. However, it is still costly to store and verify all the block headers. The cost of this solution grows linearly with the size of the blockchain.

2.3 Super Light Clients

More recently, two sublinear solutions have been proposed to reduce the cost of light clients further. Non-Interactive Proof of Proof of Work (NIPOPOW) [15] and FlyClient [11] use several randomly chosen blocks from the entire blockchain, to verify the validity of a predicate based on that blockchain. Compared to SPV, they have a lower cost for verification; the user is required to download a polylogarithmic fraction of blocks to store and verify. The user interacts with some full nodes and compares the provided proofs by each full node together to find the best proof.

In both solutions, the primary assumption is that there exists at least one honest full node among all of them. Both mechanisms detect and reject the fake chains produced by an adversary corrupting at most 33% of the consensus participants with an overwhelming probability.

2.3.1 NIPoPoWs

NIPoPoW chooses the high-difficulty blocks, as they happen less frequently than other blocks, to be verified as a sub-chain of the underlying chain. Verifying the validity of those blocks proves that enough PoW for the underlying chain has been provided with a high probability. The blocks containing a higher difficulty than their required target are called *Superblocks*. Superblocks are rare and happen randomly. The protocol compresses a long blockchain by representing it only with the high-difficulty blocks (superblocks). To do so, they need to link superblocks together. This connectivity is called *interlinking* and is not included in Bitcoin or many of the other PoW blockchains. NIPoPoW uses velvet fork [22] to include the interlinking structure in the blockchain. Velvet forks require no rule modification in the consensus layer of the blockchain, and do not force all the miners to follow the new rules. They only upgrade part of the rules and miners are free to ignore it.

The number of superblocks is a logarithmic fraction of the whole blocks. For proving the inclusion of a predicate in the blockchain, the full nodes only need to provide a polylogarithmic fraction of blocks for the light client which makes the protocol more efficient in comparison to SPV. However, their solution has a major drawback, as the protocol only works well when the difficulty is constant, contrary to most blockchains that adjust their block difficulty when the hash rate changes.

2.3.2 FlyClient

Another solution is FlyClient which also proposes a polylogarithmic solution for the light client problem. FlyClient has an optimized approach in which binary search and random sampling are conducted to find the invalid blocks of an invalid chain. This way, the algorithm detects and discards invalid proofs. FlyClient achieves a proof size smaller than NIPoPoW's. In their solution, they use Merkle Mountain Range (MMR) which is an efficiently-updatable commitment mechanism that allows provers to commit to an entire blockchain with a small (constant-size) commitment while offering logarithmic block inclusion proofs with position binding guarantees. They extend MMR to integrate the data about the difficulty and its transition. This enables their protocol to work fine in the cases that the difficulty of the blocks gets updated (unlike NIPoPoW's solution).

FlyClient, like NIPoPoW, uses velvet fork to integrate their new structure in the blockchains. It requires the block headers to include the root of MMR commitment of all previous block headers. However, they have not conducted a comprehensive analysis on the security of their protocol under velvet fork deployment. The protocol is exposed to chain-sewing attacks under such conditions. In fact, Nemoz and Zamyatin have introduced this attack and proposed some solutions to fix it [18].

In this chapter, we covered the background needed to understand the light client problem. Also, we went through the current best existing solutions.

Chapter 3

Models and Problem Formulation

In this chapter, we explain the backbone model that we used for analysis, and the underlying assumptions for the blockchain. Then, we introduce Merkle mountain trees and briefly explain how we include them in the blockchain. After that, we define the prover and verifier model that helps to better understand and formulate the problem. At last, we introduce the notations that we have used throughout the work.

3.1 Backbone Model

Our model is based on the standard backbone model for PoW blockchains [13]. This model consists of three main players: *full nodes*, *miners* and *light clients*. Full nodes maintain a copy of the whole blockchain. They validate every data they receive. Also, some full nodes relay data through the network. Miners create new blocks extending the longest chain of the blockchain and commit new transactions they receive from the clients. Miners compete to become the next block proposer by solving a puzzle which is known as *Proof-of-Work*. For getting blocks and transactions, miners rely on full nodes. Clients need up-to-date information from the blockchain. For example, when they want to perform a transaction inclusion verification, they need the block header of the desired block, in addition to a proof for the transaction inclusion in that block header. Light clients are clients who utilize fewer resources such as storage and computation. They send queries to some

full nodes to obtain their desired data instead of paying for the cost of maintaining the whole blockchain.

Block creation is captured in the random oracle model as in [13]. This model has a security parameter σ. In each mining round, a miner sends q queries to the random oracle function $H(\cdot)$. For each query, if the value has not been queried before, the function returns a random value from $\{0,1\}^{\sigma}$ and stores the input and output values in a table. If the value has been queried before, the function finds the recorded output from the table and returns it.

For a block to be valid, it needs to have a proof to show enough work has been done on that block. Each block B has a target difficulty T. If $H(B) < T$, we say the block meets its difficulty requirement, therefore, it has a sufficient amount of work. The target difficulty T is set in a way that the block intervals remain constant in expectation. We call the expectation of block intervals *Block Interval Time*. In our model, the target difficulty of the blocks can be constant or variable.

We abstract B to contain the block header information. A valid block that has a valid PoW and extends the longest chain will be propagated by honest nodes and with a high probability will get finalized in the blockchain. A block gets finalized when it is buried under a chain of at least k other blocks, where k is the *finality parameter*. The finality parameter k is tuned in a way that the probability of a finalized block slipping out of the longest chain is made negligible. The finality parameter differs from one blockchain to another. For example, for the Bitcoin ledger, we know that $k = 6$.

Each block has a block header. Every block header contains a block number, the hash of the previous block as a pointer to it, the target difficulty T, the Merkle root of all transactions, and some other information related to the block. The size of block headers is much smaller than the size of the blocks themselves. Mostly because the block header does not contain the transactions themselves.

When a miner receives a new block, that block may extend some block other than the last one and cause a fork in that miner's local blockchain. Honest nodes will follow the longest chain rule. They choose the longest valid chain and start mining their new block on it. In a valid chain, all the blocks meet their difficulty requirement. We assume more than half of the mining power in the network is honest. Any blockchain that an honest node maintains is called an *honest chain*.

3.2 Chain Commitment

We leverage the notion of *Merkle Mountain Range (MMR)* in our protocol to enable verifying data from any previous block header of the blockchain. MMR is a more efficient variant of Merkle trees [16]. For the MMR root of each block, the MMR leaves are the block headers of all previous blocks. Just like Merkle trees, parent nodes in MMR are hash values of their two children. Using MMR, the entire blockchain can be committed into a single hash value. Also, the proof of the inclusion of a block in the MMR tree is of logarithmic size. MMR uses an efficient updating process to append new leaves and update its root. Each block's MMR root can be constructed from the previous block's MMR root with a low computation overhead. It is also capable of efficiently removing the last leaf from the structure. Moreover, the consistency of the MMR root of a past block with the MMR root of a more recent block can be shown easily. The process of adding a leaf to an MMR tree is shown in Figure 3.1. Since an MMR tree might have more than one peak, the MMR root is constructed from bagged peaks.

For blockchains that do not have MMR root in their block header structure, an upgrade to the consensus layer is required for the structure to include MMR root in it. One possible way for upgrading is to do a hard fork or a soft fork [8]. If doing such a fork is not possible, a velvet fork, introduced by Kiayias et al. in [15], can be used to add MMR root to the block structure. The changes they propose in the velvet fork need no rule modifications to the consensus layer. Honest miners will be divided into two groups of upgraded and non-upgraded honest miners. The upgraded miners are required to include the MMR root of the previous blocks in their coinbase data, however, non-upgraded miners just ignore that data as comments. Coinbase is the first transaction of a block, determined by the block's miner. Both

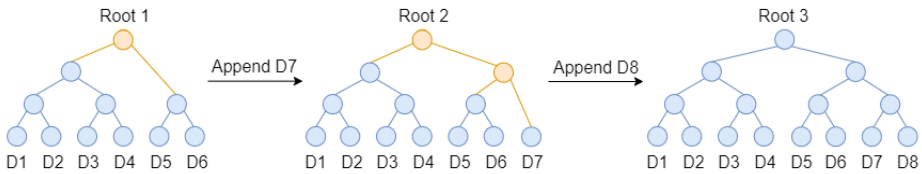

Figure 3.1: Merkle Mountain Range

types of miners accept previously valid blocks regardless of whether they have upgraded structures or not.

In the velvet fork, the set of accepted blocks is not modified, therefore, there may exist some upgraded blocks containing an invalid MMR root. We assume that the majority of the upgraded miners are honest. We add a voting mechanism so that valid MMR roots can be detected. To enable the voting, we require every upgraded miner to include a buffer of length α in their coinbase data as well as the MMR root. Later, we will define α in a way for the protocol to be secure and discuss it. Every upgraded block has a flag in its coinbase data so that it can be recognized from the non-upgraded blocks.

3.3 The Prover and Verifier Model

To verify some data being included in the blockchain without having to maintain the whole blockchain, a client needs to send some queries to full nodes. We call the light client a *verifier* and the full nodes *prover*s. We assume that the verifier doesn't trust any of the provers and is attempting to find a prover to get some finalized blocks of the honest chain. This way, she can perform a state or transaction inclusion verification test and verify a piece of data is included in the blockchain. We assume that the verifier knows the genesis block, the block interval time and the confirmation number of the blocks k.

The verifier sends the same request to several provers and they will respond to her with a proof. The verifier needs to determine which of the proofs belongs to an honest prover. The proof contains a chain of blocks. We assume that there exists at least one honest full node among the selected provers, i.e., the client has not been eclipsed from the network.

Without loss of generality, we can assume that the verifier sends requests to two full nodes where one of them is honest and the other one is malicious. The verifier's goal is to determine which of them is the honest node. The honest node maintains the honest chain which is a blockchain with a specific genesis block that has maximum mining power working on it. In other words, it has the highest cumulative difficulty. The verifier wants to confirm some data from this honest chain. In real-world scenarios, the verifier can send requests to multiple provers to

increase the probability of communicating with an honest node.

3.4 Notations

We denote a chain of length n of block headers by $C[0:n-1]$. In this chain, $C[i]$ $(0 \leq i \leq n-1)$ refers to the block header with height i, and $C[i:j]$ refers to the set of block headers from height i inclusive to height j inclusive. Specifically, $C[i:end]$ is the set of block headers starting from $C[i]$ to the end of the chain.

In this chapter, we explained the models and notions that we use in our solution. We also discussed our assumptions. Now, we are ready to understand how the LIGHTSYNC protocol works.

Chapter 4

LightSync Protocol

In this chapter, we present the protocol of LIGHTSYNC. The main properties of the LIGHTSYNC protocol are twofold. First, the proposed protocol is very simple, and it can easily be implemented for any PoW blockchain without the need to change its consensus rule. Second, the computation power and the storage that a verifier needs have been dramatically reduced. In particular, the number of the blocks that a verifier needs to download from the full nodes does not depend on the length of the chain and is a constant factor.

4.1 Protocol Overview

As we discussed in the prover and verifier model in the previous chapter, we assume that we have one verifier and two provers where only one of the provers is honest.

The verifier starts the protocol by sending a query to the provers. She creates a transaction and sends it to both of the provers. We call this transaction the *query transaction*. After sending requests to the provers, the verifier waits to get responses. Every prover should respond with a proof before a predetermined deadline. The verifier compares the received proofs using LIGHTSYNC algorithm. The winner proof belongs to the honest node with an overwhelming probability.

The proof consists of two parts. First, a proof to get the last finalized block. Second, a proof to find the last valid MMR root of the blockchain. Using the first part of the proof, the verifier gets the last finalized block of the honest chain.

Furthermore, the second part of the proof provides information about the history of the blockchain in case that the verifier needs to verify some data based on the past blocks of the blockchain.

Next, we explain the details of each part of the proof.

4.2 Getting The Last Finalized Block Header

The verifier sends a *query transaction* to both provers. Provers will try to include the *query transaction* in their local blockchains and provide the verifier with a valid proof within the *challenge period* to prove the honesty of their local blockchains. The *challenge period* is determined by the verifier and is sent to the provers along with the *query transaction*. The longer the *challenge period* is, the higher certainty it provides for the verifier about its final decision. The *challenge period* starts when the verifier sends the *query transaction* to the provers. The steps of the protocol for finding the last finalized block header are described in Algorithm 1.

To include the *query transaction* in its local blockchain, each prover broadcasts it into his local blockchain network. The honest prover propagates the transaction in the whole blockchain network so that the honest mining power working on the last block of the blockchain includes this transaction in their block. After the *query transaction* is included in their local blockchains, the provers will wait for some block confirmations. The provers should send their proofs to the verifier before the *challenge period* is passed. The proof consists of a sub-chain of block headers from the blockchain, as well as a Merkle inclusion proof of the *query transaction* in one of those block headers. The sub-chain contains block headers from the block including the *query transaction* to the last mined block on top of it that gets mined before the *challenge period* is passed. Provers should include at least $k+1$ block headers in their proofs. If the number of blocks mined after the block containing the *query transaction* is less than k, the provers should include the previous block headers to contain at least $k+1$ block headers in their proof. In Figure 4.1 and Figure 4.2, the structure of the proof's block headers is provided for both cases.

The honest prover can simply wait for the blockchain to grow and have some block confirmations, and send the proof to the verifier a few moments before the deadline. On the other hand, for the malicious prover, the honest mining power will

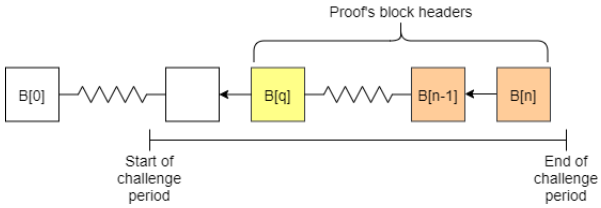

Figure 4.1: Proof provided by the prover for finding the last finalized block ($m > k+1$)

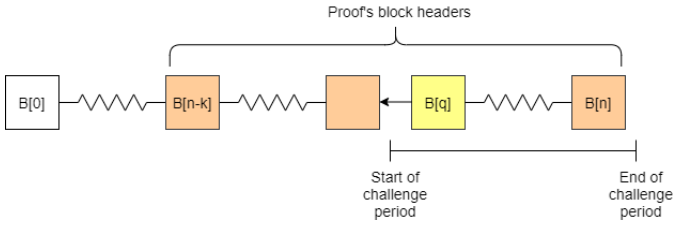

Figure 4.2: Proof provided by the prover for finding the last finalized block ($m < k+1$)

not help him to grow his chain, as long as his local chain data contradicts the honest chain data. The malicious prover, therefore, has no more power than the adversary mining power to mine confirmation blocks on top of the block containing the *query transaction*. We call the confirmation block headers along with the block header including the *query transaction*, the *challenge headers*. Since the adversary mining power is less than the total honest mining power due to the underlying blockchain's security, in chapter 5, we show that the *overall difficulty* of the *challenge headers* provided by the malicious prover in the *challenge period* is less than the honest prover's with very high probability.

Definition 1. *The overall difficulty of a group of blocks is the summation of the inverse of the target difficulty of each block.*

The greater a block's target difficulty is, the easier it would be to build that block. Therefore, overall difficulty is a measure of the actual difficulty of creating a group of blocks. Also, we know that building more difficult blocks requires more computing power. So, overall difficulty is a great measure to see which group of

blocks had more computation power working on it during the same time period.

The verifier receives all the proofs provided within the *challenge period*. Note that both the verifier and the provers should be online so that the verifier can examine whether the proof was provided within the *challenge period* or not. Each proof consists of a chain of block headers. First, the verifier checks the inclusion of her *query transaction* in one of the block headers of the proof using the Merkle proof of inclusion. Then, she checks the validation of the rest of the block headers. Each block header should refer to its previous block header correctly and have valid PoW regarding its target difficulty. If a proof passes all the above checks, we consider it as a *valid proof*. In the next step, the verifier decides which proof to accept by calculating all valid proofs' *overall difficulties* and comparing them together. The valid proof with the highest *overall difficulty* gets accepted. We call this proof the *winner proof*.

Algorithm 1 LightSync Protocol for Finding the Last Finalized Block Header

1: The verifier creates a query transaction and sends it to the selected provers along with the challenge period.
2: The verifier starts a timer for the challenge period.
3: Each prover who receives the query transaction runs CREATEPROOF(*queryTransaction*) (Procedure 2) and sends the created proof to the verifier.
4: For each received proof, the verifier runs VALIDATEPROOF(*proof*) (Procedure 3).
5: The verifier runs OVERALLDIFFICULTY(*proof*, *queryTransaction*) (Procedure 4) for the valid proofs.
6: The verifier chooses the valid proof with the highest overall difficulty as the winner proof.

Since the malicious prover wants the verifier to accept his local chain, he tries to maximize the *overall difficulty* of his proof. It means that he tries to mine as many blocks with high difficulties as possible, on top of the block containing the *query transaction*. As long as the *query transaction* gets determined by the verifier, the provers cannot start constructing the proof before they get the *query transaction* from the verifier. Therefore, they only have a limited time (*challenge period*) to construct the proof. During this limited time, there is a negligible chance that the

Procedure 2 CREATEPROOF(*queryTransaction*)

Start a timer for the challenge period.
$\delta \leftarrow$ the communication delay between the verifier and the prover
Checks the validity of $queryTransaction$
Broadcast $queryTransaction$ in the local blockchain
repeat
    On the local longest chain $B[0 : end]$, find the block B[q] which includes $queryTransaction$
    $MerkleProof \leftarrow$ proof of inclusion of $queryTransaction$ in the $B[q]$
    $m \leftarrow len(B[q : end])$
    if $m \geq k+1$ **then**
        $C[0 : m-1] \leftarrow B[q : end]$
    else
        $C[0 : k] \leftarrow B[q-k+m : end]$
    end if
until 2δ seconds remaining from the challenge period
$proof \leftarrow C \cup MerkleProof$
return $proof$

Procedure 3 VALIDATEPROOF(*proof*)

Check the length of the chain in the $proof$ to be at least $k+1$.
Verify the correctness of Merkle inclusion proof for the query transaction.
Check that each block header of the chain refers correctly to its previous block header.
Calculate the PoW for each block header of the chain and validate it against the target difficulty.
return $true$ if all the above checks pass

Procedure 4 OVERALLDIFFICULTY(*proof, queryTransaction*)

Find the block header $C[q]$ containing $queryTransaction$
$T[i]$:= target difficulty of block header $C[i]$ $(q \leq i \leq end)$
$overallDifficulty \leftarrow \sum_{i=q}^{end} T[i]^{-1}$
return $overallDifficulty$

malicious prover could achieve a higher *overall difficulty* than the honest prover, because, the honest mining power is more than the adversary mining power in the network.

The detailed analysis of the protocol is described in chapter 5. After running the above protocol, the verifier has the last finalized block header of the blockchain and can decide on a predicate based on the recent history of the blockchain. In the following, we explain how the verifier confirms data from the history of the blockchain.

4.3 Finding The Last Valid MMR Root

After executing the first part of the LIGHTSYNC protocol, a verifier has got a copy of at least $k+1$ block headers of the blockchain. If the construction of the block headers of the blockchain includes the MMR root of all previous block headers of the blockchain, then knowing the last $k+1$ block headers, the client has access to the MMR root in a finalized block header of the blockchain. The client can perform a transaction inclusion test for transactions related to the past blocks of the blockchain using that MMR root. This is the case for some blockchains like Beam [1] and Grin [4] that include MMR root in their block header structure.

However, if a blockchain does not support MMR, there exist three possible approaches for including MMR roots in the structure and using them: Hard fork, Soft fork, or Velvet fork. In the case of a hard fork, miners are required to include an MMR root in the block headers of the blockchain. Alternatively, in a soft fork, the MMR root is added to the new blocks in a way to stay backward compatible with the old blocks. For example, the MMR root can be included in a predetermined transaction like the coinbase transaction. This way the non-upgraded miners will accept the new blocks whether these blocks are created following the new rule or not. Nonetheless, the upgraded miners will follow the new rule and only accept blocks including a valid MMR root. A soft fork needs the majority of the miners to upgrade to the new rule. After including the MMR root in the structure of the block headers of a blockchain using either of the above two approaches, it can be used immediately to verify data from the history of the blockchain by the verifier. For example, Zcash [20] has added MMR root to its block header structure using a

hard fork named Heartwood.

The last approach is to use a velvet fork to add MMR roots to the block headers. In velvet forks, there is no need for a specific number of miners to upgrade. The upgraded miners will add the MMR root to the block headers in a backward-compatible way (like soft forks it can be included in the coinbase transaction data), but, they continue to accept blocks created by non-upgraded miners. This way the set of accepted blocks stays unchanged, meaning that every non-upgraded or upgraded miner will accept a valid block, whether it includes an MMR root or not. In this case, since no one is checking the validity of the MMR field, some of the blocks may contain invalid MMR roots. If the verifier accepts an invalid MMR root, the malicious prover has deceived the verifier to accept a predicate that contradicts the honest chain. To avoid invalid MMR roots getting accepted by the light clients, we append a small buffer of data to the MMR root that enables a voting mechanism for them. Applying this change, the upgrade remains backward compatible. Using this mechanism, we make sure that the verifier can detect a valid MMR root.

The buffer's length is α. Each bit of the buffer points to a preceding block and votes for its MMR root validity. In a block header, the last bit of its voting buffer refers to the last upgraded block, the bit before that refers to the second last upgraded block, and so on. The miner sets a bit to 1 (*accept vote*) if he believes that the corresponding block includes a valid MMR root and sets it to 0 (*reject vote*) otherwise. We assume that the majority of the upgraded miners are honest. This way if the majority of miners in a long enough sub-chain have voted 1 for an MMR root, the verifier can make sure that MMR root is valid and by adding new block headers to it, she can easily construct the MMR root of the last finalized block by herself.

The second part of the proof is sent by the prover to the verifier in the case of a velvet fork, to provide her with the latest valid MMR root. The verifier starts the second part of the protocol by sending the last finalized block header and a parameter called β to the prover. In response to the verifier's request, the prover sends a sub-chain of block headers that includes $\alpha+\beta$ upgraded block headers, as well as each block's coinbase data with proof of its inclusion in the block. We call the first β block headers of the sub-chain *candidates*. In chapter 5, we discuss how

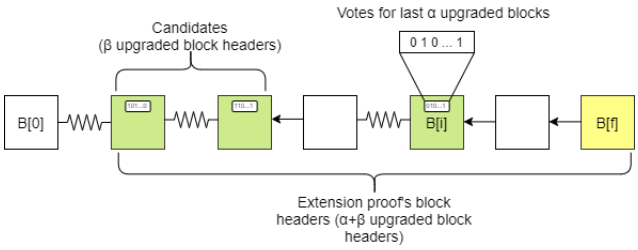

Figure 4.3: Proof provided by the prover for obtaining the last valid MMR, in case of using a velvet fork

β effects the protocol security. Algorithm 5 shows the steps of the second part of the protocol. In Figure 4.3, the structure of the proof's block headers is provided.

First, the verifier validates the received proof. She checks the previous hashes of the block headers to be correct and the coinbase data inclusions to be valid. She also checks the extended proof to include the last finalized block header from the previous part. After checking the validity of the proof, the verifier extracts candidates' block headers. Each of the candidates has α voters vote for them, as long as every upgraded block header votes for previous α upgraded block headers. An MMR root among the candidates that has been accepted by more than half of its voters is a valid MMR root. The verifier will accept that MMR root and use it to obtain the last valid MMR root of the blockchain by appending the block headers that exist on top of it to its tree. By choosing the right β, there exists a valid MMR root among the candidates with an overwhelming probability. A detailed analysis of this voting mechanism and how to determine the parameters α and β is discussed in chapter 5. At last, the verifier can check the inclusion of the genesis block against the valid MMR, by asking the prover to send her a proof of inclusion, to make sure she has received the correct chain.

In this chapter, we introduced LIGHTSYNC protocol and explained its algorithm in detail. LIGHTSYNC is a low-cost and simple solution to the light client problem. Next, we will prove that it is fully secure.

Algorithm 5 LightSync Protocol for Finding the Last Valid MMR Root

1: The verifier sends an MMR request along with the last finalized block header $B[f]$ to a prover.
2: The prover creates a proof containing two parts:

   1. The last $s+1$ block headers up to $B[f]$, which is $B[f-s:f]$, that contains $\alpha+\beta$ upgraded block headers from his local chain $B[0:f]$.

   2. The coinbase data for each block header along with its proof of inclusion in that block header PROOF(*CoinbaseData*).

3: The prover sends the proof to the verifier.
4: The verifier runs FINDLASTMMR($B[f-s:f]$, $CoinbaseData[0:s]$, PROOF($CoinbaseData[0:s]$)) to find the last valid MMR root.

Procedure 6 FINDLASTMMR($C[0:s]$, $CoinbaseData[0:s]$, PROOF($CoinbaseData[0:s]$))

1: Check that $C[s]$ is equal to $B[l]$.
2: Check that the number of upgraded block headers is equal to $\alpha+\beta$.
3: Check that each block header refers correctly to its previous block header.
4: Check the correctness of each coinbase data inclusion proof.
5: $candidates[0:\beta-1] \leftarrow$ first β upgraded block headers of $C[0:s]$
6: $V[0:\beta-1]$:= the number of accept votes for each block header of $candidates[0:\beta-1]$
7: *validSet* $\leftarrow$ the *candidates*'s block headers with more than $\lfloor \alpha/2 \rfloor$ accept votes
8: *validRoot* $\leftarrow$ the MMR root of the last block header of *validSet*
9: $v \leftarrow$ the index of the block containing *validRoot*
10: *lastValidRoot* $\leftarrow$ update the *validRoot* by adding the block headers $C[v:s]$ to its tree
11: **return** *lastValidRoot*

Chapter 5

Protocol Analysis

In this chapter, we perform a thorough analysis of the security of the LIGHTSYNC protocol. We also show that it is secure under the variable difficulty settings.

As we described in the prover and verifier model, the verifier sends queries to two provers where one of them is honest and the other one is malicious. Each prover sends a proof to the verifier. After executing the protocol, the verifier decides on one of the proofs as the winner proof. The LIGHTSYNC protocol is secure if the verifier decides on the honest prover's proof with an overwhelming probability. In what follows, we explain the security conditions of the protocol and prove them.

The delay of the peer-to-peer network is so small in comparison to the block interval time that it can be approximated and assumed to be zero. However, the whole analysis can be performed again with the assumption of the network delay being greater than zero. In that case, the effective computation power of the honest nodes will decrease. Because some of the honest miners will get the latest data with a delay, their computation power will get wasted during that time. So, more honest computing power will be needed in the network to reach the same security we had without considering the network delay.

We assume that the majority of the underlying blockchain miners are honest which is a necessary condition for Proof-of-Work blockchains. In the case where MMR structure has been added to the blockchain using velvet fork (the approach described in section 4.3), we assume that the majority of the upgraded miners are

honest.

Regarding the difficulty of the blocks, we assume that in the blocks that are sent as the first part of the proof, the difficulty of the blocks would change at most one time. This is a realistic assumption as long as the difficulty adjustment in a blockchain gets done in longer time intervals than the *challenge period*. For example, difficulty adjustment period is 2 weeks for Bitcoin.

The malicious prover wants to deceive the verifier and make her confirm some predicate that contradicts the honest chain data. The verifier validates her predicate using the last valid MMR root. So, the malicious prover should be able to convince the verifier to accept an invalid MMR root to succeed in deceiving the verifier. In the following, we analyze both cases where all the blocks include an MMR root (case I) and where the MMR root is added to some of the blocks using a velvet fork (case II). The former is the case where the blockchain includes the MMR root in its structure (and so there is no need to do a fork), also, it is the case of using a hard fork or soft fork to add MMR to the structure.

Case I. In this case where all blocks include an MMR root, the malicious prover should send an invalid block as a finalized block to be able to mislead the verifier, because, the last valid MMR root is included in the last finalized block header. If the malicious prover wants his invalid block to be the winner proof for the verifier, he has to include some confirmation blocks containing higher *overall difficulty* than the other prover. Based on Theorem 1, the probability of this scenario is negligible under our model assumptions.

The probability of the query transaction being included in the first block mined after the *challenge period* starts is determined by its transaction fee. The higher the transaction fee is, the higher will be the chance for it to be included in the blockchain sooner. We assume the transaction fee to be high enough so that the query transaction gets included in the first mined block after the challenge period starts. If the transaction fee is not high enough, the transaction will be included in later block headers of the honest chain, however, the adversary mining power will include it in the first block of its local chain anyways. Hence, there will be a higher chance for the malicious proof to have more overall difficulty and win.

The proof includes $m \geq k+1$ ($m' \geq k+1$) block headers, for the honest (malicious) prover. We denote this sub-chain by $C[0:m-1]$ ($C'[0:m'-1]$). The *query*

transaction is included in one of the block headers of the proof, suppose the block header to be $C[q]$ ($C'[q']$). The verifier calculates the *overall difficulty* of $C[q : end]$ and $C'[q' : end]$. The proof with the higher *overall difficulty* will be chosen as the *winner proof*.

The target difficulty for each of these two sub-chains can change at most one time. We denote the number of block headers using the same target difficulty T_1 (T_1') which are constructed in time t_1 (t_1') starting from the beginning of the *challenge period* by random variable N_1 (N_1'). In the remaining time t_2 (t_2'), blocks are constructed with target difficulty T_2 (T_2'). We denote the number of blocks constructed during this time with random variable N_2 (N_2'). Real numbers t_1, t_2, t_1', and t_2' are non-negative where $t_1 + t_2$ and $t_1' + t_2'$ are equal to the *challenge period* (t).

Denote the honest and the adversary mining powers with h and a respectively. We can see that the total number of hashes taken to create N_1 blocks is $t_1 h$. The probability of each hash leading to a valid Proof of Work is $T_1/2^\kappa$ (where κ is the security parameter used in [13]). This means that the discrete random variable N_1 has a binomial distribution with parameters $(n = t_1 h, p = T_1/2^\kappa)$ for any given t_1. The same conclusion can be made for N_2, N_1', and N_2'. So, the discrete random variables N_2, N_1', and N_2' have binomial distributions with parameters $(n = t_2 h, p = T_2/2^\kappa)$, $(n = t_1' a, p = T_1'/2^\kappa)$, and $(n = t_2' a, p = T_2'/2^\kappa)$, for any given t_2, t_1', and t_2', respectively.

Since the *challenge period* is much less than the difficulty adjustment time interval, we can assume that the whole mining power of the network is constant during this time. All the blocks of the adversary chain should meet their target difficulties, otherwise, the verifier would not accept their proof as a valid proof. However, the adversary mining power will not necessarily use the target difficulties of the honest chain. They may use other target difficulties in order to maximize their chance of being selected as the winner proof.

To prove the security of the algorithm, let's first prove two lemmas, then we will get to Theorem 1 and its proof.

Lemma 1. *Suppose that N_1' has a binomial distribution with parameters $n = t_1' a$, and $p = p_1' = T_1'/2^\kappa$. We show that*

$$\begin{aligned}&\Pr\{N_1' > n_1\frac{T_1'}{T_1} + n_2\frac{T_1'}{T_2} - n_2'\frac{T_1'}{T_2'}\}\\ &\leq \frac{(1+p_1'(e^m-1))^{t_1' a p_1'}}{(e^{mn_1\frac{T_1'}{T_1}})(e^{mn_2\frac{T_1'}{T_2}})(e^{-mn_2'\frac{T_1'}{T_2'}})}\end{aligned} \tag{5.1}$$

where $m, T_1, T_2, T_1', T_2' > 0$ *and* $n_1, n_1', n_2, n_2' \geq 0$.

Proof. For some $m > 0$, using the Chernoff bound we have:

$$\begin{aligned}&Pr\{N_1' > n_1\frac{T_1'}{T_1} + n_2\frac{T_1'}{T_2} - n_2'\frac{T_1'}{T_2'}\}\\ &\leq \frac{\mathbb{E}[e^{mN_1'}]}{e^{mn_1\frac{T_1'}{T_1} + mn_2\frac{T_1'}{T_2} - mn_2'\frac{T_1'}{T_2'}}}\end{aligned} \tag{5.2}$$

Now, notice

$$\mathbb{E}[e^{mN_1'}] = \sum_{n_1'=0}^{\infty} e^{mn_1'} Pr\{N_1' = n_1'\} = (1+p_1'(e^m-1))^{t_1' a p_1'} \tag{5.3}$$

and the result follows. □

Lemma 2. *Suppose that* $h, a > 0$ *and* $h > a$. *Function* $f(m, T, T')$ *is defined as:*

$$\begin{aligned}f(m,T,T') = &(1+T/2^{\kappa}(e^{-m/T}-1))^{h/a}\\ &\times(1+T'/2^{\kappa}(e^{m/T'}-1)).\end{aligned} \tag{5.4}$$

Then, for every $T, T' > 0$, *there exists some* $m > 0$ *such that* $f(m, T, T') < 1$.

Proof. Calculating derivative of $f(m, T, T')$ with respect to m, we get

$$\frac{\mathbf{d}f(m,T,T')}{\mathbf{d}m} = \frac{1}{2^{\kappa}}\left(1 - \frac{h}{a}\right) < 0 \tag{5.5}$$

for $m = 0$. Knowing $f(0, T, T') = 1$, we can conclude that for small enough $m > 0$, we have $f(m, T, T') < 1$.

□

We know the verifier sends a query transaction (denoted by *tx*) to two provers where exactly one of them is honest. Denote the proof that the honest prover provides by *proof(t)*, and the proof that the malicious prover provides by *proof'(t)* (proofs are functions of the challenge period denoted by t). The verifier and the honest prover are running the LIGHTSYNC protocol, however, the malicious prover acts as he wants.

Theorem 1. *Assume the* OVERALLDIFFICULTY *function from Definition 1. Then,*

$$\lim_{t\to\infty} \Pr\{\text{OVERALLDIFFICULTY}(proof'(t), tx) > \text{OVERALLDIFFICULTY}(proof(t), tx)\} = 0 \quad (5.6)$$

Proof. Using the notations we defined in this chapter, we know that:

$$\text{OVERALLDIFFICULTY}(proof(t), tx) = \frac{N_1}{T_1} + \frac{N_2}{T_2} \quad (5.7)$$

and

$$\text{OVERALLDIFFICULTY}(proof'(t), tx) = \frac{N_1'}{T_1'} + \frac{N_2'}{T_2'}. \quad (5.8)$$

So, the probability (1) is equal to

$$\begin{aligned} &\Pr\left\{N_1' > N_1\frac{T_1'}{T_1} + N_2\frac{T_1'}{T_2} - N_2'\frac{T_1'}{T_2'}\right\} \\ &= \sum_{n_1=0}^{\infty}\sum_{n_2=0}^{\infty}\sum_{n_2'=0}^{\infty} \Pr\left\{N_1' > n_1\frac{T_1'}{T_1} + n_2\frac{T_1'}{T_2} - n_2'\frac{T_1'}{T_2'}\right\} \\ &\quad \times \Pr\{N_1 = n_1\}\Pr\{N_2 = n_2\}\Pr\{N_2' = n_2'\}. \end{aligned} \quad (5.9)$$

Now using Lemma 1 and the fact that N_1, N_2, and N_2' have binomial distributions, we see that (5.9) is upper bounded by

$$\leq \sum_{n_1=0}^{\infty}\left(\frac{\binom{t_1 h}{n_1} p_1^{n_1}(1-p_1)^{t_1 h - n_1}}{e^{mn_1\frac{T_1'}{T_1}}}\right)$$

$$\times \sum_{n_2=0}^{\infty} \left(\frac{\binom{t_2 h}{n_2} p_2^{n_2} (1-p_2)^{t_2 h - n_2}}{e^{m n_2 \frac{T_1'}{T_2}}} \right)$$

$$\times \sum_{n_2'=0}^{\infty} \left(\frac{\binom{t_2' a}{n_2'} p_2^{\prime n_2'} (1-p_2')^{t_2' a - n_2'}}{e^{-m n_2' \frac{T_1'}{T_2'}}} \right) \times \left(1 + p_1'(e^m - 1)\right)^{t_1' a p_1'} \tag{5.10}$$

which is equal to

$$\begin{aligned} &(1 + p_1(e^{-mT_1'/T_1} - 1))^{t_1 h p_1} \times (1 + p_1'(e^m - 1))^{t_1' a p_1'} \\ &\times (1 + p_2(e^{-mT_1'/T_2} - 1))^{t_2 h p_2} \times (1 + p_2'(e^{mT_1'/T_2'} - 1))^{t_2' a p_2'} \end{aligned} \tag{5.11}$$

for any $m > 0$. Here, $p_1 = T_1/2^\kappa$, $p_2 = T_2/2^\kappa$, $p_1' = T_1'/2^\kappa$, and $p_2' = T_2'/2^\kappa$.

Since $t_1 + t_2 = t_1' + t_2'$, and due to symmetry, we have two possibilities for the order of these parameters: $t_1 < t_1' < t_2' < t_2$ or $t_1' < t_1 < t_2 < t_2'$. Without loss of generality we can assume the former to be true (in the other case, the proof is exactly the same). Define $t_0 = t_1' - t_1 = t_2 - t_2'$. Using the definition of $f(m, T, T')$ in Lemma 2, we can re-write (5.10) as

$$f(mT_1', T_1, T_1')^{at_1} f(mT_1', T_2, T_2')^{at_2'} f(mT_1', T_2, T_1')^{at_0}. \tag{5.12}$$

Using Lemma 2, we see that for a large enough t (which is the *challenge period*), the above expression can be arbitrarily small. □

Case II. In this case, the probability of the verifier getting an invalid finalized block in the first part of the protocol is similar to the previous case which we proved is negligible. In the second part of the protocol, where the verifier aims to find the last valid MMR root, for the protocol to be secure, we need to prove two important statements:

- The *candidates* contain a block header including a valid MMR root with an overwhelming probability.
- After running the second part of the protocol, the verifier will decide on a block header among the *candidates* including a valid MMR root with an overwhelming probability.

Given that a block mined by an upgraded honest node includes a valid MMR root, Theorem 2 results in the first statement. Based on Theorem 3, the probability of an MMR root that is inconsistent with the honest chain to be chosen as valid MMR root by the verifier is negligible when α is chosen properly. Parameter α is the size of the voting buffer introduced in the LIGHTSYNC algorithm. It can be deduced that the probability of a valid MMR root getting rejected by the majority of the voters is the same and is negligible. Therefore, the second statement is concluded. To give an example for a proper α, let's assume a blockchain's network where $a/(a+h) = \frac{1}{3}$ and let $\alpha = 80$. Now, based on Equation (5.13), the probability of a wrong MMR root being chosen by the verifier is less than 0.01.

Suppose the proportion of the upgraded honest mining power to be M_h and the portion of adversary mining power to be M_a. We have $M_h + M_a = 1$. We assume that $M_h > M_a$. Also, suppose the number of the block headers of *candidates* to be β.

Theorem 2. *The probability of none of the candidates' block headers having been mined by an honest node goes to zero when β goes to infinity.*

Proof. It suffices to only consider the private double-spend attack in which the adversary races with the honest nodes to grow a longer chain [12]. The *candidates* are β block headers where each of them could have been mined by the honest or the adversary mining power. The probability of a block being mined by an honest miner is M_h. We know that the probability of blocks being mined by honest nodes is independent of each other. Therefore, the probability of the *candidates* not including any honest block headers is ${M_a}^{\beta}$ which is negligible for a large enough β. □

To illustrate this probability, let us consider an example here for a blockchain network where $M_a = \frac{1}{3}$. Letting $\beta = 7$, the above probability will be less than 0.0005.

Theorem 3. *The probability of an inconsistent MMR root (R_{ic}) getting more than $\lceil \frac{\alpha}{2} \rceil$ accept votes goes to zero, when α goes to infinity.*

Proof. As R_{ic} is not consistent with the honest chain, the honest miners cast reject votes for it. At most, all the malicious miners cast accept vote for R_{ic}. The number

of malicious miners in the next α upgraded blocks is the binomial random variable X with parameters $p = M_a$ and $n = \alpha$. The probability of X being greater than or equal to $\lceil \frac{\alpha}{2} \rceil$ is:

$$\begin{aligned} \Pr\left(X \geq \lceil \frac{\alpha}{2} \rceil\right) &= \sum_{i=\lceil \frac{\alpha}{2} \rceil}^{\alpha} \binom{\alpha}{i} (M_a^i)(M_h^{\alpha-i}) \\ &= M_a^{\lceil \frac{\alpha}{2} \rceil} \sum_{i=\lceil \frac{\alpha}{2} \rceil}^{\alpha} \binom{\alpha}{i} (M_a^{i-\lceil \frac{\alpha}{2} \rceil})(M_h^{\alpha-i}) \\ < M_a^{\lceil \frac{\alpha}{2} \rceil} \sum_{i=\lceil \frac{\alpha}{2} \rceil}^{\alpha} \binom{\alpha}{i} (M_h^{\lfloor \frac{\alpha}{2} \rfloor}) &\leq (M_a M_h)^{\frac{\alpha-1}{2}} \times 2^{\alpha-1} \end{aligned} \tag{5.13}$$

We know that $M_a + M_h = 1 \wedge M_a < M_h \Rightarrow M_a M_h < \frac{1}{4} \Rightarrow M_a M_h = 2^{-(2+s)}$, where $s > 0$. Therefore, $\Pr(X \geq \lceil \frac{\alpha}{2} \rceil) < 2^{-s\frac{(\alpha-1)}{2}}$, where $s = -\log_2(M_a M_h) - 2 > 0$. So, $\lim_{\alpha \to \infty} \Pr(X \geq \lceil \frac{\alpha}{2} \rceil) = 0$. □

In this chapter, we performed a thorough analysis of the security of the LIGHT-SYNC protocol. We proved that with a constant proof size, a verifier can securely check the inclusion of some data in a Proof-of-Work blockchain using the LIGHT-SYNC protocol tuned with correct parameters.

Chapter 6

Evaluation

In this chapter, we perform a thorough analysis of the complexity of the LIGHTSYNC protocol. We compare LIGHTSYNC to the existing solutions in terms of needed resources. We also show the practicality of our solution by presenting the results of LIGHTSYNC implementation for Ethereum.

6.1 Comparison With NIPoPoW and FlyClient

The main goal of a light client protocol is to enable light clients to verify the inclusion of some data in the blockchain using the least possible storage and computation resources.

In the first part of the protocol (4.2), each prover has a limited time (*challenge period*) to provide the verifier with a proof. In LIGHTSYNC, the *challenge period* is determined by the desired level of security and is independent of the blockchain's length. The number of blocks mined in the honest chain during this period is on average the *challenge period* divided by the block interval time of the blockchain, which stays constant. If MMR root is included in the block structure using a hard fork or a soft fork, after running the first part of the protocol, no more data is required from the prover. So, the verifier validates her predicate on the honest chain using constant resources.

If MMR structure has been added to the blockchain using velvet fork, after running the first part of the protocol, more data is required to validate the predicate.

Table 6.1: Comparison of different light client protocols

Protocol	Protocol Complexity for Verifier	Added Structure
SPV	$O(n)$	-
NIPoPoW	$O(\text{polylog}(n))$	Interlink
FlyClient	$O(\text{polylog}(n))$	MMR
LightSync	$O(1)$	MMR

The winner prover sends a sub-chain that includes $\alpha+\beta$ upgraded block headers to the verifier. Assuming that the ratio of the total mining power to the upgraded mining power is l, in a long run, $1/l$ of total mined blocks of the honest chain are upgraded. So, on average, the winner prover needs to send $(\alpha+\beta)\times l$ block headers to the verifier to provide $\alpha+\beta$ upgraded block headers. Since α, β, and l do not depend on the length of the blockchain, the needed resources to perform the second part of the protocol (4.3) stays constant.

In conclusion, no matter what the size of the blockchain is, the verifier can perform data inclusion verification using constant storage and computation resources. In Table 6.1, we make a comparison between existing solutions and LIGHTSYNC. The verifier's complexity in the table refers to the needed storage and computation resources for the verifier. Although we have included FlyClient in the table, we should mention that an attack named "chain-sewing" attack [18] can be performed against this protocol, when it is under velvet fork deployment. Some countermeasures have been proposed to ensure the security of its deployment under velvet fork condition. Each of them suffer from some drawbacks, including making the protocol interactive, and increasing the cost of the protocol drastically. So, by FlyClient we don't mean the naive implementation of it which is insecure. Instead, we consider a secure modified version of it, like the interactive method.

6.2 Implementation

We implement and evaluate LIGHTSYNC using data from Ethereum blockchain which is a popular PoW blockchain. Ethereum's block header size is much larger than Bitcoin's. Also, Ethereum's blockchain is growing much faster. Currently, Ethereum has more than 15 million blocks. So, an efficient protocol for light clients

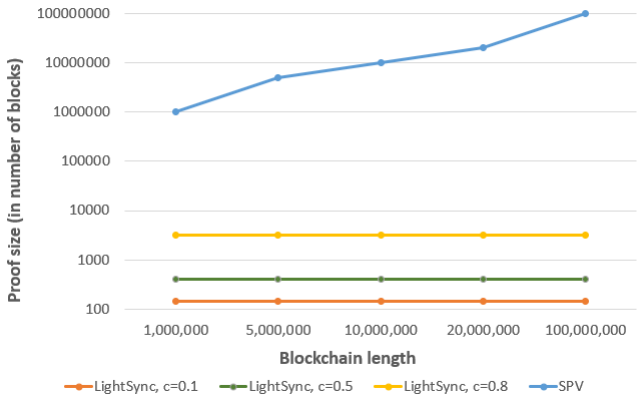

Figure 6.1: LightSync proof size in comparison with SPV

of this blockchain seem to be crucial. As we discussed before, the LIGHTSYNC proof size doesn't change with the length of the blockchain. We demonstrate the efficiency of LIGHTSYNC in Figure 6.1.

In this figure, we show the proof size in case of different adversary mining power portions. We assume the security parameter to be $\sigma = 50$, which assures the probability of failure to be less than 2^{-50}. Parameter c is the ratio of the adversary mining power to the honest mining power in the network. We can see that even for a very large c ($c = 0.8$ means that the total mining power that the adversary controls is 44.4% of the whole mining power), LIGHTSYNC extremely improves the proof size. Here, we have assumed that all the miners are upgraded. Now, in the case where velvet fork is implemented and some portion of miners have been upgraded, we can see the relation between the proof size and the ratio of upgraded miners in Figure 6.2. We have assumed $c = 0.5$ for this case.

In this chapter, we discussed the results of implementing LIGHTSYNC and evaluated its performance. The results validate the efficiency of our approach, and its functionality under velvet fork deployment.

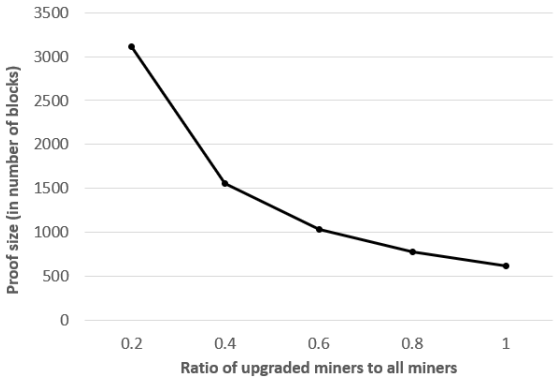

Figure 6.2: LightSync proof size vs. the ratio of upgraded miners in velvet fork for c=0.5

Chapter 7

Discussion

7.1 Staying Up-to-Date

In some applications, after running the LIGHTSYNC protocol and getting the last finalized block header and the last valid MMR root of the blockchain, the verifier may want to receive the future block headers as soon as they are mined. In this case, the verifier can continue requesting block headers from the honest full node and check their validity and compliance with her current chain of block headers and add them to her local chain if the received data is valid. When a new block header gets finalized, the verifier can add it to the MMR root she maintains to achieve the last stable MMR root of the blockchain. Then, she can drop all the finalized block headers prior to it. In this way, she will always maintain a limited number of block headers; a sub-chain with length $k+1$ with some possible forks. However, she can perform any transaction inclusion or state verification by using the MMR root she maintains.

In other cases where the verifier doesn't need to learn about new block headers immediately as they are mined, she can repeat the LIGHTSYNC protocol in long time intervals. In this way, instead of getting each block header in real-time, the client can update herself periodically. So, she will use fewer resources, but, will experience some delays.

7.2 Incentive Design

Full nodes maintain a copy of the whole blockchain and relay data in the network. In addition, they provide the verifiers with light client services which have some extra costs for them. The growing number of light clients makes the services' cost considerable for full nodes. Without a proper incentive mechanism, there is a lack of motivation for users to participate in the network as full nodes. In this section, we discuss an incentive mechanism, which proposes a way to cover the costs of being a full node and prevents the light clients from free-riding. The previous light client solutions suffer from the lack of incentive in their protocols. With the increase of demand for light clients, there is a need to properly incentivize full nodes, so that they continue providing the verifiers with light client services.

Incentive Mechanism for LIGHTSYNC. To start the protocol, the verifier sends a *query transaction* to provers. This transaction is a Pay-To-Script-Hash (P2SH)[5]: a special type of transaction that transfers money from the sender of the transaction to the hash of a script which is determined by the sender. The script may require some conditions to be satisfied. Anyone who provides the script and satisfies the dictated conditions can spend the P2SH transaction. The verifier uses this kind of transaction to pay the prover for the service she gets. We call this fee the *Service Fee* that compensates for the costs of the full nodes. The verifier also has to pay the transaction fee of the P2SH transaction.

After executing the LIGHTSYNC protocol, the verifier sends the script to the prover who has provided the final chosen proof. Using the script, the prover can spend the *query transaction* and receive his fee. The service fee makes an incentive for honest provers to participate in LIGHTSYNC and broadcast the *query transaction* in the fastest time. If more honest nodes are incentivized to participate in LIGHTSYNC, the chance of the user connecting to at least one honest prover gets higher. Also, the faster the honest prover propagates the *query transaction* in the network, the sooner an honest miner includes it in a block. A summary of all the needed communication between the verifier and the provers is shown in Figure 7.1. After receiving the proofs, the verifier detects the honest prover and continues communicating with him to obtain the MMR and provides him with the solution for conditions of P2SH transaction. We call it the puzzle of P2SH transaction.

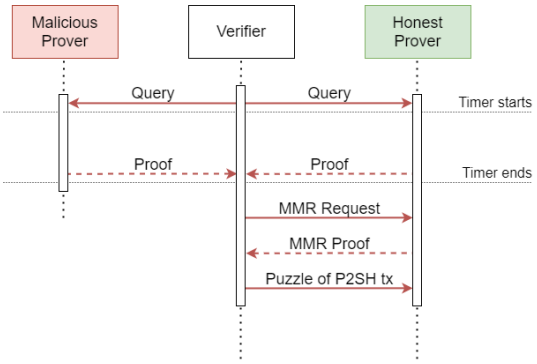

Figure 7.1: Communication between the verifier and the provers

The described mechanism has a challenge: the verifier might refuse to provide the prover with the script. In this case, although the prover has provided a service for the verifier, it hasn't received the service fee and the verifier has only paid a transaction fee. We assume that after executing the LIGHTSYNC protocol, the communication between the verifier and the prover needs to be continued. Since the verifier is a light client, she wants to receive every new block header from the prover. If the verifier refuses to reveal the script, the prover will not provide her with new block headers anymore. In this case, the verifier needs to restart the LIGHTSYNC protocol with another prover. This means that she needs to send a new *query transaction* to the other provers. This costs an extra transaction fee for her. By setting the transaction fee greater than the service fee, it will demotivate the verifier to abandon the first honest prover, so, the proposed challenge gets solved.

Chapter 8

Conclusion

In this thesis, we present LIGHTSYNC, a low-cost light client protocol that is suitable for PoW blockchains. Leveraging LIGHTSYNC, a light client can verify the inclusion of data in the blockchain using constant computation and storage resources no matter what the length of the blockchain is. We describe the details of applying LIGHTSYNC and perform a thorough security analysis of it. LIGHTSYNC can be used for lightweight devices (e.g., mobile phones and IoT devices) that intend to connect to blockchains and verify data against them. Another important application of such a protocol is in cross-chain communication, in which one blockchain wants to verify the correctness of data against the other one. For future work, we are going to extend the LIGHTSYNC protocol to support blockchains using other consensus mechanisms (e.g. Proof of Stake). Also, we will design a cross-chain bridge using the LIGHTSYNC.

Bibliography

[1] Beam description. comparison with classical mw. URL https://docs.beam.mw/BEAM_Comparison_with_classical_MW.pdf. → page 23

[2] Btc relay, a bridge between the bitcoin blockchain & ethereum smart contracts. URL http://btcrelay.org/. → page 3

[3] Ethereum chain full sync data size. URL https://ycharts.com/indicators/ethereum_chain_full_sync_data_size. → page 3

[4] Merkle mountain ranges (mmr). URL https://docs.grin.mw/wiki/chain-state/merkle-mountain-range/. → page 23

[5] Pay to script hash. URL https://en.bitcoin.it/wiki/Pay_to_script_hash. → page 40

[6] Eth-near rainbow bridge. URL https://near.org/blog/eth-near-rainbow-bridge/. → page 3

[7] R. Belchior, A. Vasconcelos, S. Guerreiro, and M. Correia. A survey on blockchain interoperability: Past, present, and future trends, 2021. → page 3

[8] V. Buterin. Hard forks, soft forks, defaults and coercion. URL https://vitalik.ca/general/2017/03/14/forks_and_markets.html. → page 15

[9] V. Buterin. Ethereum white paper: A next generation smart contract & decentralized application platform. 2014. URL https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.pdf. → page 3

[10] V. Buterin. Chain interoperability. 2016. URL https://www.r3.com/wp-content/uploads/2017/06/chain_interoperability_r3.pdf. → page 3

[11] B. Bünz, L. Kiffer, L. Luu, and M. Zamani. Flyclient: Super-light clients for cryptocurrencies. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 928–946, 2020. doi:10.1109/SP40000.2020.00049. → page 11

[12] A. Dembo, S. Kannan, E. N. Tas, D. Tse, P. Viswanath, X. Wang, and O. Zeitouni. Everything is a race and nakamoto always wins. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 859–878, 2020. → page 33

[13] J. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In J. Katz and H. Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 291–323, Cham, 2017. Springer International Publishing. ISBN 978-3-319-63688-7. → pages 13, 14, 29

[14] P. Gaži, A. Kiayias, and D. Zindros. Proof-of-stake sidechains. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 139–156. IEEE, 2019. → page 3

[15] A. Kiayias, A. Miller, and D. Zindros. Non-interactive proofs of proof-of-work. In J. Bonneau and N. Heninger, editors, *Financial Cryptography and Data Security*, pages 505–522, Cham, 2020. Springer International Publishing. ISBN 978-3-030-51280-4. → pages 11, 15

[16] R. C. Merkle. A digital signature based on a conventional encryption function. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO '87, page 369–378, Berlin, Heidelberg, 1987. Springer-Verlag. ISBN 3540187960. → page 15

[17] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2009. URL http://www.bitcoin.org/bitcoin.pdf. → pages 2, 9

[18] T. Nemoz and A. Zamyatin. On the deployment of flyclient as a velvet fork: chain-sewing attacks and countermeasures. Cryptology ePrint Archive, Report 2021/782, 2021. https://eprint.iacr.org/2021/782. → pages 12, 36

[19] S. Paavolainen and C. Carr. Security properties of light clients on the ethereum blockchain. *IEEE Access*, 8:124339–124358, 2020. doi:10.1109/ACCESS.2020.3006113. → page 5

[20] Y. Tong Lai, J. Prestwich, and G. Konstantopoulos. Flyclient - consensus-layer changes. 2019. URL https://zips.z.cash/zip-0221. → page 23

[21] A. Zamyatin, M. Al-Bassam, D. Zindros, E. Kokoris-Kogias, P. Moreno-Sanchez, A. Kiayias, and W. Knottenbelt. Sok: Communication across distributed ledgers. *IACR Cryptol. ePrint Arch.*, 2019:1128, 2019. → page 3

[22] A. Zamyatin, N. Stifter, A. Judmayer, P. Schindler, E. Weippl, and W. J. Knottenbelt. A wild velvet fork appears! inclusive blockchain protocol changes in practice. In A. Zohar, I. Eyal, V. Teague, J. Clark, A. Bracciali, F. Pintore, and M. Sala, editors, *Financial Cryptography and Data Security*, pages 31–42, Berlin, Heidelberg, 2019. Springer Berlin Heidelberg. ISBN 978-3-662-58820-8. → pages 8, 11