**BRNO UNIVERSITY OF TECHNOLOGY**
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

**FACULTY OF INFORMATION TECHNOLOGY**
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

**DEPARTMENT OF INFORMATION SYSTEMS**
ÚSTAV INFORMAČNÍCH SYSTÉMŮ


# PROCESSING OF THE BLOCKCHAIN EMPLOYING IPFS

VYUŽITÍ IPFS PRO ZPRACOVÁNÍ BLOCKCHAINU

**MASTER'S THESIS**
DIPLOMOVÁ PRÁCE


**AUTHOR** **Bc. MATÚŠ MÚČKA**
AUTOR PRÁCE

**SUPERVISOR** **Ing. VLADIMÍR VESELÝ, Ph.D.**
VEDOUCÍ PRÁCE


**BRNO 2020**

## Abstract
Do tohoto odstavce bude zapsán výtah (abstrakt) práce v anglickém jazyce.



## Abstrakt
Do tohoto odstavce bude zapsán výtah (abstrakt) práce v českém (slovenském) jazyce.



## Keywords
Sem budou zapsána jednotlivá klíčová slova v anglickém jazyce, oddělená čárkami.

## Klíčová slova
Sem budou zapsána jednotlivá klíčová slova v českém (slovenském) jazyce, oddělená čárkami.



## Reference
MÚČKA, Matúš. *Processing of the Blockchain Employing IPFS*. Brno, 2020. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Vladimír Veselý, Ph.D.

# Processing of the Blockchain Employing IPFS

## Declaration

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana X... Další informace mi poskytli... Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

........................
Matúš Múčka
January 14, 2020

## Acknowledgements

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant apod.).

# Contents

# Chapter 1

# Introduction

HTTP is „good enough“ for the most use cases of distributing files over the network. However, when we want to stream lots of data to multiple connected clients at once, we are starting to hittings its limits. When two clients are requesting the same data, there is no mechanism in HTTP that would allow sending the data only once. Sending duplicate data has become a problem in large companies because of bandwidth capacity. Blizzard [1] started to distribute video game content by distributed solution because it was cheaper for the company and faster for players [4]. Linux distributions use BitTorrent to transmit disk images [2].

The bitcoin blockchain has now 242 gigabytes [3]. When blockchain is processed (parsed address, created search indexes), the size on a disk can doubles. If there are multiple blockchains, then data can have few terabytes. When we are sharing blockchains data from the server for several clients, there is a big chance that multiple clients want the same data. They may be working on the same case and investigating the same wallets. So in standard solution with relational database and some HTTP server, for every request server have to search in all data (that can have a size of few terabytes) and transmit selected data to the client. This problem happens even if a different client asks for the same data in a few minutes ago. Behaviour mentioned above dramatically limits the scalability of the server.

Services that are using HTTP, often have client-server architecture, so there is also a problem with one point of failure. If the server for some reason stops working, the client can not receive data. In distributed file system such as IPFS, there is no such problem as one point of failure, because all data are duplicated on multiple clients.

**[[Structure of the paper]]**

[1] Game company

[2] Image of Debian downloadable by BitTorrent https://www.debian.org/CD/torrent-cd/

[3] Current size of bitcoin blockchain can be seen at https://www.statista.com/statistics/647523/worldwide-bitcoin-blockchain-size/

# Chapter 2

# Cryptocurrencies

There were hundreds of failed attempts of creating cryptographic payment systems before cryptocurrencies like Bitcoin and Ethereum come into existence. Some of these systems are listed in figure 2.1. All of them were created before Bitcoin, and despite that, some of these attempts were only academic proposals, others were deployed and tested systems, only a few of them survived to these days. One of the survival is PayPal, but only because it quickly give up its original idea of hand-held devices for cryptographic payments.[9]

So there is a question, what makes cryptocurrencies successful nowadays? It may be easy to use principle and no need for external hardware. Another critical component of cryptocurrencies discussed in this work is Blockchain. Generally, it is a ledger in which are all transactions securely stored. The idea behind blockchains is pretty old, and it was originally used for timestamping digital documents.[3]

| | | | | |
|---|---|---|---|---|
| ACC | CyberCents | iKP | MPTP | Proton |
| Agora | CyberCoin | IMB-MP | Net900 | Redi-Charge |
| AIMP | CyberGold | InterCoin | NetBill | S/PAY |
| Allopass | DigiGold | Ipin | NetCard | Sandia Lab E-Cash |
| b-money | Digital Silk Road | Javien | NetCash | Secure Courier |
| BankNet | e-Comm | Karma | NetCheque | Semopo |
| Bitbit | E-Gold | LotteryTickets | NetFare | SET |
| Bitgold | Ecash | Lucre | No3rd | SET2Go |
| Bitpass | eCharge | MagicMoney | One Click Charge | SubScrip |
| C-SET | eCoin | Mandate | PayMe | Trivnet |
| CAFÉ | Edd | MicroMint | PayNet | TUB |
| CheckFree | eVend | Micromoney | PayPal | Twitpay |
| ClickandBuy | First Virtual | MilliCent | PaySafeCard | VeriFone |
| ClickShare | FSTC Electronic Check | Mini-Pay | PayTrust | VisaCash |
| CommerceNet | Geldkarte | Minitix | PayWord | Wallie |
| CommercePOINT | Globe Left | MobileMoney | Peppercoin | Way2Pay |
| CommerceSTAGE | Hashcash | Mojo | PhoneTicks | WorldPay |
| Cybank | HINDE | Mollie | Playspan | X-Pay |
| CyberCash | iBill | Mondex | Polling | |

Figure 2.1: Electronic payment systems before cryptocurrencies [7]

## 2.1 Bitcoin

Bitcoin is probably the most famous cryptocurrency. On 3 January 2009, Satoshi Nakamoto (an alias for a person or group persons authored the bitcoin white paper) mined the genesis block of bitcoin (block with height 0). Satoshi gets the reward of 50 bitcoins (half a million US dollars in the time of writing). This text was embedded in the block(see 2.2): *The Times 03/Jan/2009 Chancellor on brink of second bailout for banks.*[1] [2].

```
00000000   01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00000010   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00000020   00 00 00 00 3B A3 ED FD  7A 7B 12 B2 7A C7 2C 3E   ....;£íýz{.²zÇ,>
00000030   67 76 8F 61 7F C8 1B C3  88 8A 51 32 3A 9F B8 AA   gv.a.È.Ã^ŠQ2:Ÿ¸ª
00000040   4B 1E 5E 4A 29 AB 5F 49  FF FF 00 1D 1D AC 2B 7C   K.^J)«_Iÿÿ...¬+|
00000050   01 01 00 00 00 01 00 00  00 00 00 00 00 00 00 00   ................
00000060   00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ................
00000070   00 00 00 00 00 00 FF FF  FF FF 4D 04 FF FF 00 1D   ......ÿÿÿÿM.ÿÿ..
00000080   01 04 45 54 68 65 20 54  69 6D 65 73 20 30 33 2F   ..EThe Times 03/
00000090   4A 61 6E 2F 32 30 30 39  20 43 68 61 6E 63 65 6C   Jan/2009 Chancel
000000A0   6C 6F 72 20 6F 6E 20 62  72 69 6E 6B 20 6F 66 20   lor on brink of 
000000B0   73 65 63 6F 6E 64 20 62  61 69 6C 6F 75 74 20 66   second bailout f
000000C0   6F 72 20 62 61 6E 6B 73  FF FF FF FF 01 00 F2 05   or banksÿÿÿÿ..ò.
000000D0   2A 01 00 00 00 43 41 04  67 8A FD B0 FE 55 48 27   *....CA.gŠý°þUH'
000000E0   19 67 F1 A6 71 30 B7 10  5C D6 A8 28 E0 39 09 A6   .gñ¦q0·.\Ö¨(à9.¦
000000F0   79 62 E0 EA 1F 61 DE B6  49 F6 BC 3F 4C EF 38 C4   ybàê.aÞ¶Iö¼?Lï8Ä
00000100   F3 55 04 E5 1E C1 12 DE  5C 38 4D F7 BA 0B 8D 57   óU.å.Á.Þ\8M÷°..W
00000110   8A 4C 70 2B 6B F1 1D 5F  AC 00 00 00 00            ŠLp+kñ._¬....
```

Figure 2.2: Genesis bitcoin block

First documents purchase happened in 22nd May 2010, when Laszlo Hanyecz bought two pizzas for 10 000 bitcoins ($41 then, now about $80 000 000)[2]. This transactions hash is `a1075db55d416d3ca199f55b6084e2115b9345e16c5cf302fc80e9d5fbf5d48d` and will be stored in bitcoin blockchain forever. Actual photo of $80 000 000 pizza is on figure 2.3.

## 2.2 DigiByte

DigiByte was developed and released in 2013. It is based on Bitcoin with some adjustment in the code to improving functionality. In late 2017 there was 200 000 pending transaction in Bitcoin. Miners preferred transaction with a bigger fee, so to confirm a transaction, user needed to pay $50. Digibyte solved this problem by adding a new block every 15 seconds (new block in Bitcoin is mined every 10 minutes). Average transaction occupies 570 bytes of data. One black can contain approximately 3 500 transactions given the 2 MB limit. This means that in DigiByte, 500 transactions can be confirmed in one second compared to Bitcoin 4-7 transaction per second. DigiByte also has 1000:1 DigiByte to Bitcoin ratio so for every Bitcoin there will be 1000 DigiByte.

[1] https://en.bitcoin.it/wiki/Genesis_block
[2] https://bitcointalk.org/index.php?topic=137.0


Figure 2.3: Pizza for 10 000BTC

## 2.3 Ethereum

While both the Bitcoin and Ethereum networks are powered by the principle of distributed ledgers and cryptography, the two differ technically in many ways. For example, transactions on the Ethereum network may contain executable code, while data affixed to Bitcoin network transactions are generally only for keeping notes. Other differences include block time (an ether transaction is confirmed in seconds compared to minutes for bitcoin) and the algorithms that they run on (Ethereum uses ethash while Bitcoin uses SHA-256).

## 2.4 Decred

Decred is cryptocurrency build from Bitcoin. Main difference from Bitcoin is the rewarding system from mining. In Bitcoin, the miner gets full reward for a mined block. Sometimes, the Bitcoin blockchain splits when two or more miners found a block at nearly the same time. The fork is resolved when the subsequent block(s) are added, and one of the chains becomes longer than the alternative(s). In Decred the chance of blockchain forks is minimized by hybrid proof of work and proof of state system. Each time a block is created (by a miner), it is not automatically part of the blockchain. Block needs to be approved by ticket holders. Then miner receives a block reward (newly created DCR). If the block is rejected by ticket holders, the miner does not receive a reward. Tickets holders for a new block are chosen randomly. The ticket validates the previous block. A block needs at least three of the five votes chosen to approve it for it to be validated. This hybrid system has many implications, including making a 51% hashpower attack very difficult, and making a minority fork very difficult as well.

## 2.5 Monero

Three years after Bitcoin, in 2012, the competing Bytecoin cryptocurrency entered the market. The problem with this cryptocurrency, however, was that 80% of all coins were mined in advance by its authors. The chances of mining were, therefore, not balanced. This led to the decision that this cryptocurrency would start again. It saw the light of day 18 April 2014 and was called BitMonero, a composite of the word coin in Esperanto (Monero) and Bitcoin according to Bitcoin. However, after five days, the community decided to use only Monero for short.

# Chapter 3

# IPFS

IPFS stands for InterPlanetary File System and is a peer-to-peer distributed filesystem designed to make the Web faster, safer, and more open. In contrast with standard filesystems, objects in IPFS are content-addressed, by the cryptographic hash of their contents. In the case of the standard Web, when user wants some file, he needs to know on which server is a file located and the full path to the file (see figure 3.1). In IPFS user needs only to know the hash of the requested file. He does not care about the location of the file (see figure 3.2). Let us take a very famous file, and add it to IPFS. Here is the plain-text MIT license:

```
Permission is hereby granted, free of charge, to any person obtaining a
copy of this software and associated documentation files (the „Software"),
to deal in the Software without restriction, including without limitation
the rights to use, copy, modify, merge, publish, distribute, sublicense,
and/or sell copies of the Software, and to permit persons to whom the Soft-
ware is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED „AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM DAMAGES OR OTHER LIA-
BILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
THE SOFTWARE.
```

If somebody tries to add this license as a file to IPFS, it will return `QmWpvK4bYR7k9b1feM48fskt2XsZfMaPfNnFxdbhJHw7QJ` every time. That is now, and will be in the future, the *content address* of that file. Later, when user try to get this file by its hash, he can get it from a random person that added it into IPFS in the past.

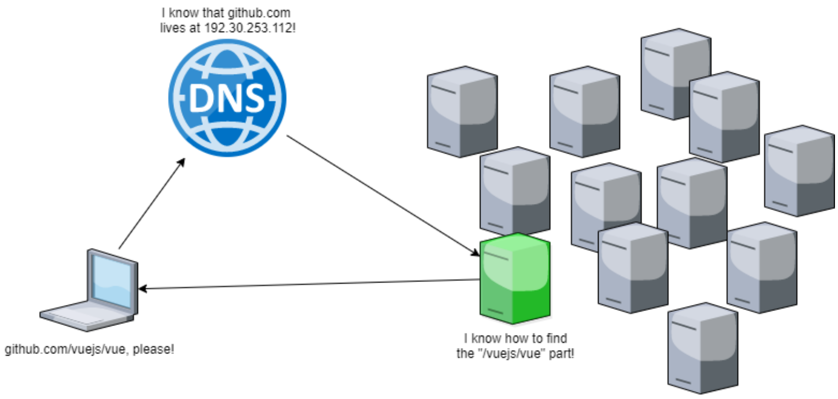


Figure 3.1: Classic web addresing

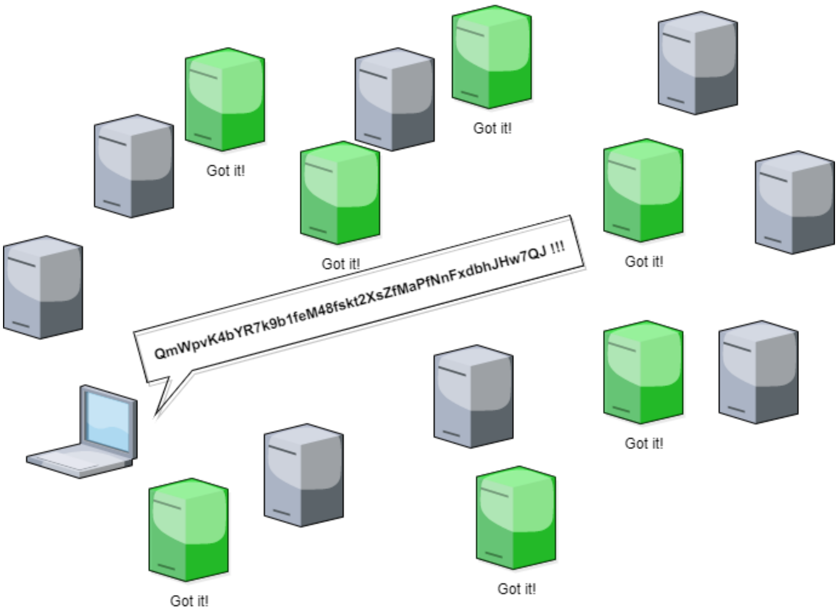


Figure 3.2: Content-based addressing

## 3.1 IPFS stack

Like in other network models, we can IPFS split into layers (see figure 3.3). There is *libp2p*[1] at the bottom, which is peer-to-peer networking module, that handles peer and content discovery, transport, security, identity, peer routing, and messaging. *IPLD* is the data model of the content-addressable web. It is providing linking between objects and multihash computing. On the top is *IPFS* which allows to publish and share files (or any data).[1]

[1] https://libp2p.io/

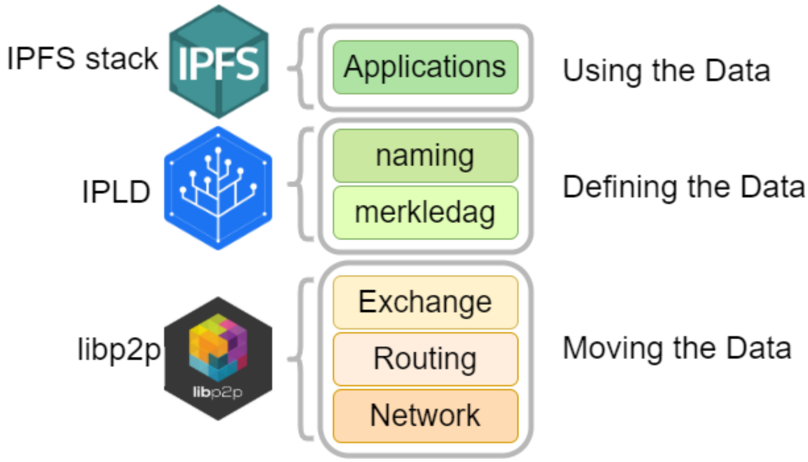


Figure 3.3: IPFS stack

## 3.2 libp2p

Libp2p is transporting layer for IPFS. It grew out of IPFS to solve networking problems in p2p networks, but now it does not require or depend on IPFS. Today many projects use libp2p as their network transporting layer. For both content discovery and peer routing, libp2p uses Kademlia-based distributed hash table. With Kademlia libp2p iteratively route requests closer to the desired peer or content using Kademlia routing algorithm [6]. In the future, Kademlia might be changed easily to some other solutions that implement simple interface in figure 3.4.

```
// gets a particular peer's network address
FindPeer(node NodeId)

// stores a small metadata value in DHT
SetValue(key []bytes, value []bytes)

// retrieves small metadata value from DHT
GetValue(key []bytes)

// announces this node can serve a large value
ProvideValue(key Multihash)

// gets a number of peers serving a large value
FindValuePeers(key Multihash, min int)
```

Figure 3.4: libp2p interface

| Property | | Value |
|---|---|---|
| Multibase | | base58btc |
| Version | | cidv0 |
| Multicodec | | dag-pb |
| Multihash | Hash Type | sha2-256 |
| | Hash Length | 256 |
| | Hash | 7e1b666c0327...3dc3022f |

Table 3.1: Human redable version of CID `QmWpvK4bYR7k9b1feM48fskt2XsZfMaPfNnF-xdbhJHw7QJ`

## 3.3 IPLD

IPLD is providing linking and addressing objects with CID (Content ID). CID is hash-based self-describing content identifier (usually encoded to base58[2] format) which includes codec and multihash. Multihash is then further composed of hashtype and hash value. Let us look closer on the MIT license file, that we add to IPFS at the beginning of this chapter (see 3). It's CID is `QmWpvK4bYR7k9b1feM48fskt2XsZfMaPfNnFxdbhJHw7QJ`. It can be converted to human-readable format as can be seen on figure 3.1, thanks to multicodec table[3]. We can see that this CID is encoded in base58 format and the file was stored using protobuf[4] codec (this information is necessary to decode file correctly).

## 3.4 IPFS

IPFS is the top layer from the IPFS stack. It is used for pinning objects and files, naming system (see IPNS[5]) and keys management. File or object is automatically pinned when a user adds it (but other IPFS commands do not include automatic pinning). Pinning a CID tells an IPFS server that the data is important and must not be thrown away. When garbage collection is triggered on a node, any pinned content is automatically exempt from deletion. Non-pinned data may be deleted. The InterPlanetary Name System (IPNS) is a system for creating and updating mutable links to IPFS content. Since objects in IPFS are content addressed, an object's address changes every time an object's content changes. A name in IPNS is the hash of a public key. It is associated with a record containing information about the hash it links to that is signed by the corresponding private key.

## 3.5 Existing blockchain explorers in IPFS

**[[TODOOOO]]**

[2] https://en.wikipedia.org/wiki/Base58
[3] https://github.com/multiformats/multicodec/blob/master/table.csv
[4] https://en.wikipedia.org/wiki/Protocol_Buffers
[5] https://docs.ipfs.io/guides/concepts/ipns/

# Chapter 4

# Design

The system consists of one or more Feeders and Explorers. Feeders are connected to external API and provide synchronization between cryptocurrency and IPFS data. Explorer can request data from the system network and display them to user.

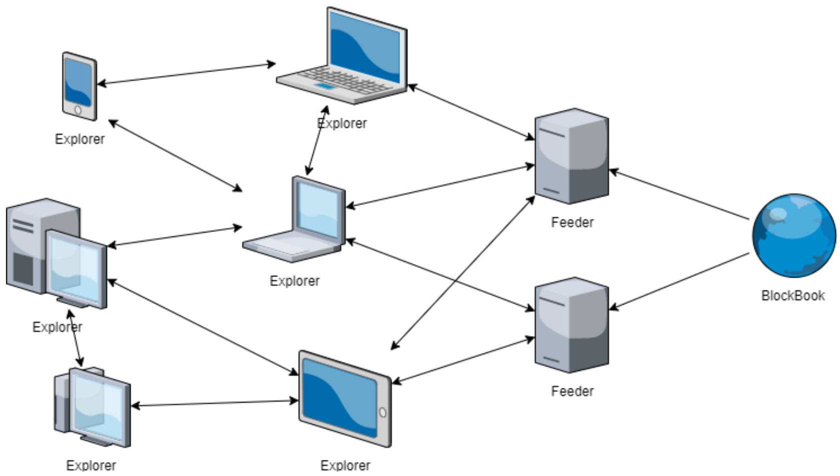


Figure 4.1: System design

### 4.0.1 Blockbook

Blockbook[1] is a blockchain indexer for Trezor Wallet[2], developed by SatoshiLabs[3]. It currently supports more than 30 coins (and some others were implemented by the community). For data storage Blockbook is using RocksDB[4] developed by Facebook, which is a NoSql database which stores only key-value pairs. Blockbook is providing fast API for accessing blocks, addresses and transactions. Main limitations of blockbook:

- **Not distributed** (client-server architecture) - problem with scalling for more users.
- **Not a SQL database** - it does not have a relational data model, it does not support SQL queries, and it has no support for indexes.

[1] https://github.com/trezor/blockbook
[2] https://wallet.trezor.io/
[3] https://satoshilabs.com/
[4] https://github.com/facebook/rocksdb/wiki

- **Single-Process** - only a single process (possibly multi-threaded) can access a particular database at a time.

## 4.1 Feeder

A Feeder is a command-line application that and stores data in IPFS for all cryptocurrencies specified in the config file. For obtaining data, it uses Blockbook API. Feeder stores data in structure such as in figure 4.2. Each block (except genesis and last block) has a link to the previous and next block. Also, it has links to transactions that had been processed in this block. A transaction has links to address and previous/spent transaction for every input/output. This scheme allows store blockchain data in IPFS in small objects with size less than 256kb (a limit for storing objects directly in DHT). Also, every logical link between objects is preserved.

The feeder should create indexes as it stores objects. These indexes will help Explorer perform queries.

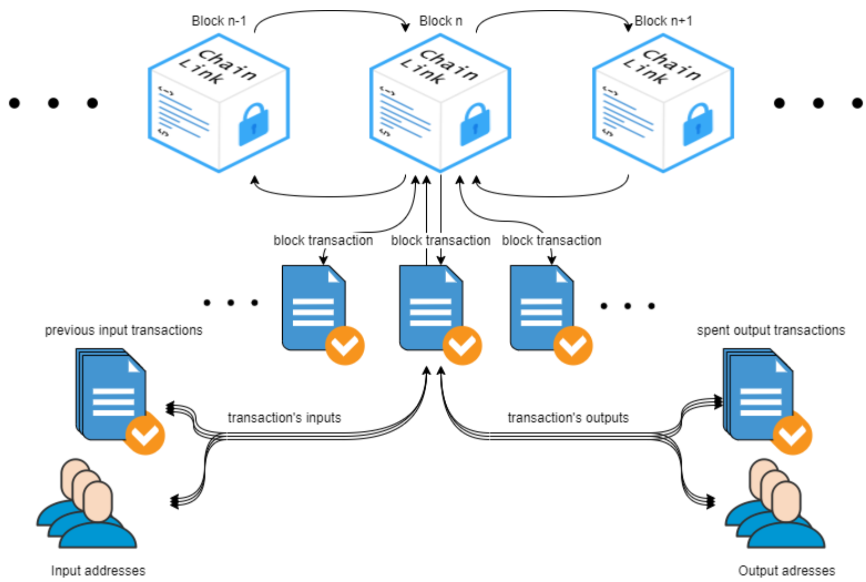


Figure 4.2: Feeder data storage structure

## 4.2 Explorer

Explorer is an application providing simple GUI and API. Application is runnable in browser or Node.js[5]. Explorer can perform basic queries like a search for block by its height or hash and search address and transaction by hash. Nevertheless, depending on specified indexes in Feeder, Explorer can also make more complex queries, for example, get 20 transaction where the sum of inputs is more than 0.5, or get transactions between some time interval.

[5]https://nodejs.org/

# Chapter 5

# Implementation

Both Feeder and Explorer are implemented in Typescript, and they are using js-ipfs[1] implementation of IPFS node. This environment selection allows code sharing between these two separated applications.

## 5.1 Feeder implementation

Informations about supported cryptocurrencies and enabled indexes are stored in Fedder's config file. Based on these settings, Feeder after start will begin to downloading data from Blockbook API, save them to IPFS, and create indexes of it.

### 5.1.1 Indexes

In the prototype, there are implemented a few different ways of indexing data in IPFS.

- **OrbitDB**[2] is a serverless, distributed, peer-to-peer database build on top of IPFS, developed by HAJA networks[3]. OrbitDB is a good solution for small user's databases, but it is still in the alpha stage of developing, and it is not well optimized to store hundreds of gigabytes of data. The biggest problem is that OrbitDB performs all queries locally. To perform query like `db.query((tx) => tx.amount > 0.001)` OrbitDB needs to load all database locally and then cycling between them. So every client ends up with a full copy of the database. This limitation is not usable for our case when we have a database that has hundreds of gigabytes of data. [5]

- **Textile**[4] is a set of open-source tools that provide a decentralized database, remote storage, user management, and more, over the IPFS network. Textile already created applications for storing photos[5], notes[6] or anything else[7]. Textile provides a high abstraction on top of the IPFS and provides simple API to store and index files securely. It uses *Cafe* peers to provides backups and indexing. Every data store is duplicated on several *Cafe* peers. When a client is obtaining some data, it will

[1] https://github.com/ipfs/js-ipfs
[2] https://orbitdb.org/
[3] https://haja.io/
[4] https://textile.io/
[5] https://textile.photos/
[6] https://noet.io/
[7] https://anytype.io/

contact one of the *Cafe* peers, and *Cafe* peer will resolve a query for the client. This is a problem for our solution because using textile require lots of harddisk memory and does not solve the problem with overloading *Cafe* peers. [8]

After some research, I came to the conclusion that currently, there is no solution for storing and indexing data in IPFS without high harddisk memory consummation. So I created my own indexing system that currently supports three types of indexes.

- **Dictionary** - a simple key-value structure that can be used for translating (for example, block height to block). Search complexity is `O(1)`, which is the fastest achievable speed. Big disadvantage is that client needs to download a whole dictionary to performs search. In the time of writing this thesis, Ethereum has 9 250 000 blocks. If we want to make dictionary for translating block height to IPFS block address, the size of this dictionary would be at least `(int_size + multihash_size) * 9 250 000` where minimal size for `int_size` is `4` bytes and `multihash_size` is `36` bytes when sha-256 is used (32 bytes) and multihash prefix is 4 bytes long. So this dictionary would have over 1.3 GB only for Ethereum. Another disadvantage is the impossibility to performing range search (for example get blocks between 9 249 950 and 2 500 000).

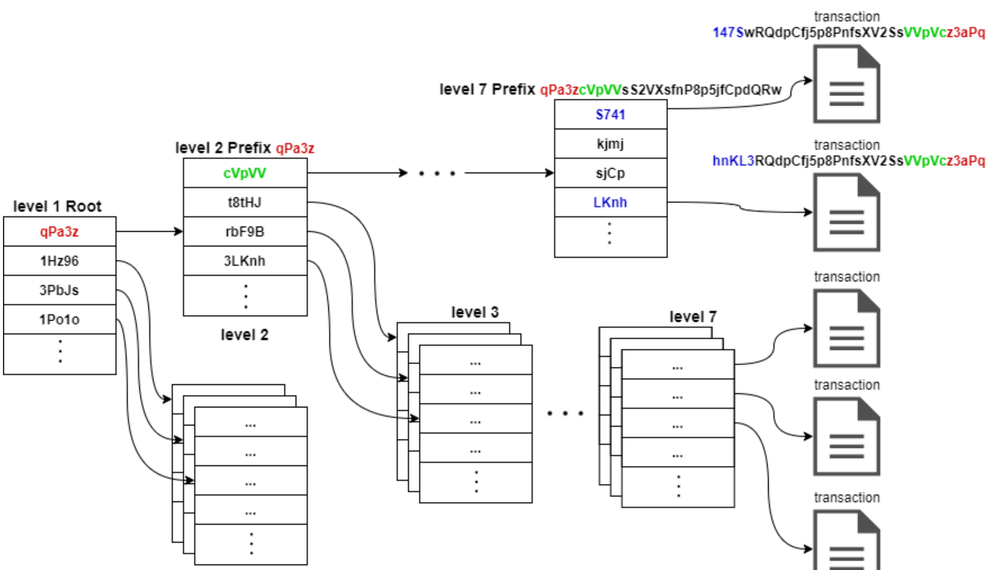


Figure 5.1: Reverse lookup by transaction hash

- **Reverse lookup** - this structure is inspired by Reverse DNS lookup[8] and it's principle can be seen on figure 5.1. For every item that will be stored in this index, the key is reversed (for better selectivity) and split into the smaller substrings. Leading substring of every key is stored in root (level 1) dictionary and is pointing to another dictionary that consists of following substrings which have the corresponding prefix in parent dictionary. Last level dictionary substrings are pointing to IPFS objects. In this index, there is a problem with performing range select operations, because items have reversed key.

[8] https://en.wikipedia.org/wiki/Reverse_DNS_lookup

- **B+ Tree** - perhaps most powerfull index structure. With auto balanced B+ tree, we can efficiently search objects by given key and performs range selects. Example of this structure can be seen in figure 5.2. By default, there is the limit of 32 items in one node (but can be changed from 4 to 256).

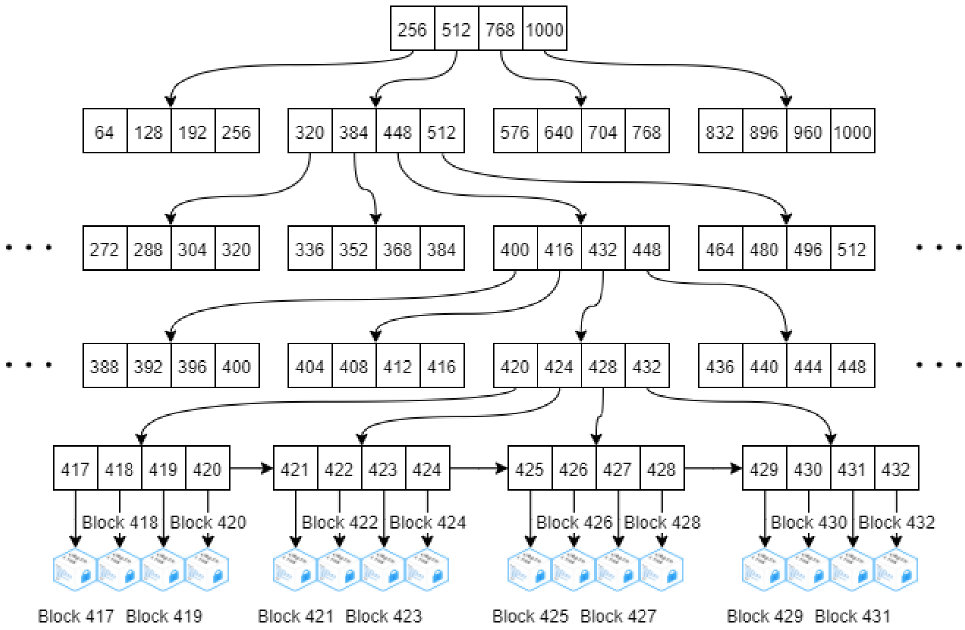


Figure 5.2: B+ tree index structure

Implementing these indexes in IPFS was surprisingly simple thanks to IPLD objects and links.

## 5.2 Explorer implementation

Currently, there are two options for running Explorer. First one is running Explorer like web application in a browser and the second one is to use Explorer as RestAPI. This architecture is shown in figure 5.3.

### 5.2.1 Running in browser as web aplication

Running in browser with GUI module provides simple user interface implemented as a single page application. Browser's implementation of IndexedDB is used as a storage for IPFS as can be seen in figure 5.4. Communication with other peers is provided though WebRTC[9] or WebSockets. Every tab opened in the browser is the same IPFS node. Opening a new tab in Incognito mode or different browser will spawn different IPFS node.

### 5.2.2 Running in node.js as REST api

Node.js[10] implementation of IPFS uses filesystem to store data (figure 5.5). On the top of ExplorerCore, there is a simple web framework Express[11], where are routes (endpoints)

[9] https://webrtc.org/
[10] https://nodejs.org/
[11] http://expressjs.com/

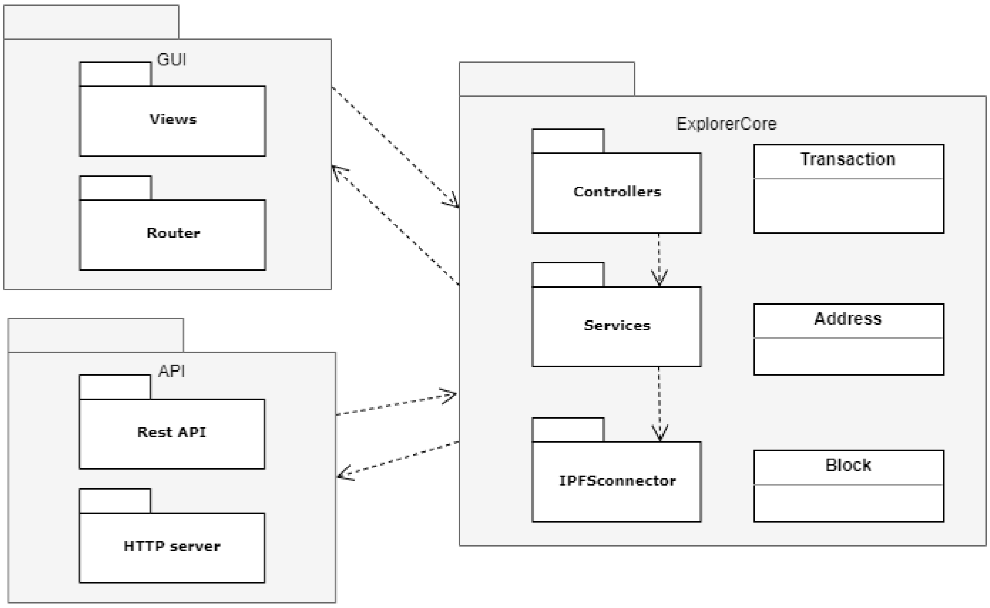


Figure 5.3: Explorer architecture

defined. Currently supported routes are described in table 5.1. Every route supports query parameters `filter` uses for filtering results and `limit` which limits number of results. Pagination can be made by setting `filter` to be greater/smaller (depending on ordering) as key of the last displayed object and `limit` to page size. For example, if a user is looking at page of transactions ordered by time (ordered from the newest transactions to the oldest), the next page of transactions are first $N$ transactions that happened before last displayed transaction (where $N$ is page size).

Rest API supports optional path param/s `path` that is useful for traversing objects in IPFS. If we want to get fifth transaction of the block with height 1000 one of the way is request URL `/block/998/next_block/next_block/txs/5` (get block with height 998, get next block two times, get transaction, a get fifth item from array of transactions). This approach allows the user to explore IPFS storage as graph very quickly by objects links.

| Endpoint | Description |
|---|---|
| `/tx/{txHash}/{path*}` | Get transaction by hash |
| `/block/{blockHeight}/{path*}` | Get block by height |
| `/block/{blockHash}/{path*}` | Get block by hash |
| `/address/{addressHash}/{path*}` | Get address by hash |

Table 5.1: Rest API explorer endpoints

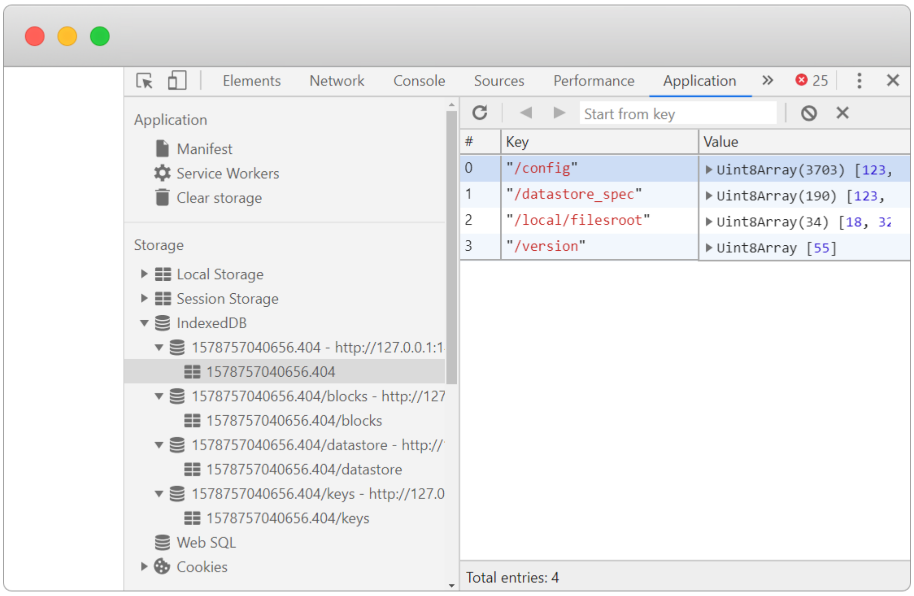


Figure 5.4: IPFS storage in browser

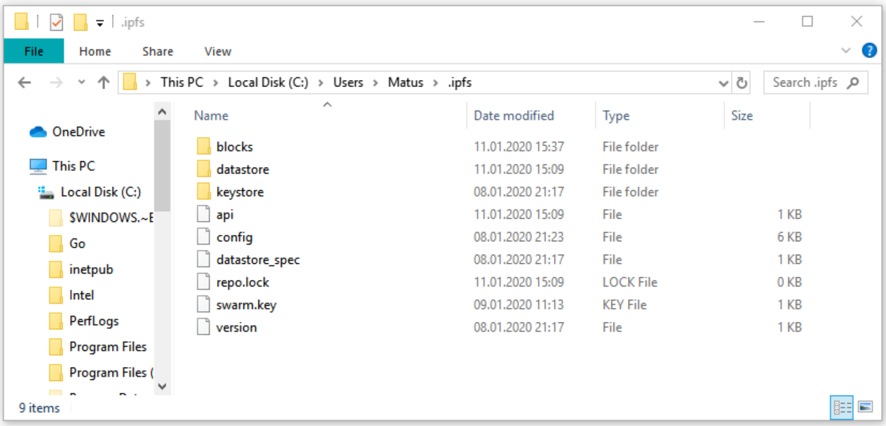


Figure 5.5: IPFS storage in node.js

# Chapter 6

# Conclusion

[[TODOOOO]]

# Bibliography


[1] BENET, J. IPFS - Content Addressed, Versioned, P2P File System. *CoRR*. 2014, abs/1407.3561. Available at: http://arxiv.org/abs/1407.3561.

[2] DAVIS, J. The Crypto-Currency. *The New Yorker*. october 2011. Available at: https://www.newyorker.com/magazine/2011/10/10/the-crypto-currency.

[3] HABER, S. and STORNETTA, W. S. How to time-stamp a digital document. In: Springer. *Conference on the Theory and Application of Cryptography*. 1990, p. 437–455.

[4] KALLE, K. *Big data in video games*. Lappeenranta, FI, 2017. Bachalar Thesis. Lappeenranta University of Technology, School of Business and Management, Computer Science. Available at: http://lutpub.lut.fi/handle/10024/147666.

[5] MARK ROBERT HENDERSON, S. P. *The OrbitDB Field Manual*. 2019. Available at: https://github.com/orbitdb/field-manual.

[6] MAYMOUNKOV, P. and MAZIERES, D. Kademlia: A peer-to-peer information system based on the xor metric. In: Springer. *International Workshop on Peer-to-Peer Systems*. 2002, p. 53–65.

[7] NARAYANAN, A., BONNEAU, J., FELTEN, E., MILLER, A. and GOLDFEDER, S. *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016.

[8] PICK, S. and HAGOPIAN. *A protocol and event-sourced database for decentralized user-siloed data*. 2019. Available at: https://blog.textile.io/introducing-textiles-threads-protocol/.

[9] WAYNER, P. *Digital cash: Commerce on the net*. Academic Press Professional, Inc., 1997.