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 11, 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 (like webpages, etc.). But 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 [2]. 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 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.

[1] Game company

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

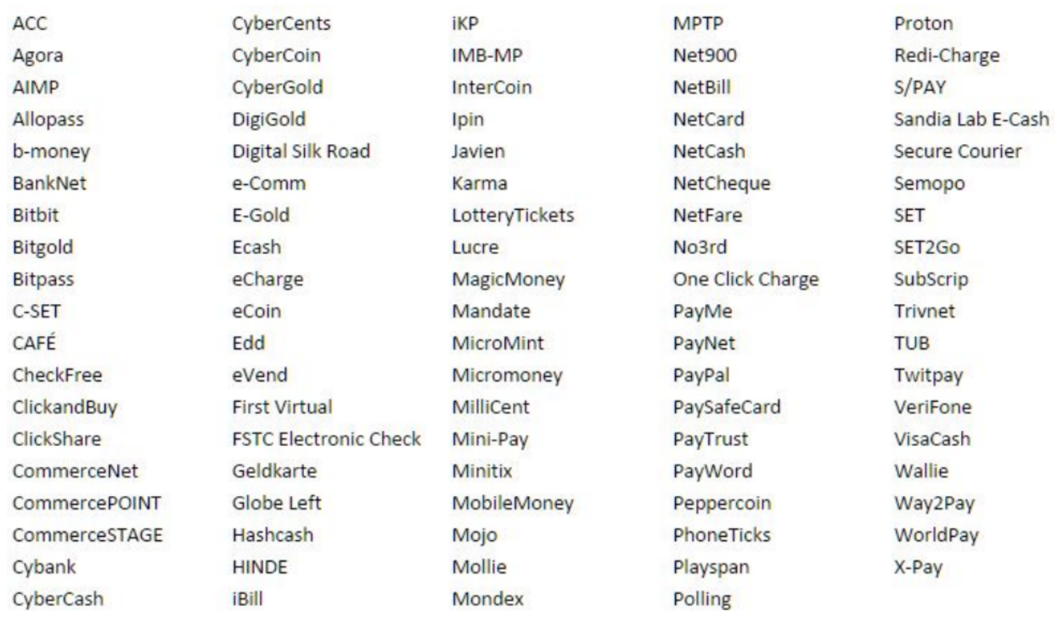
[3] Current size of bitcoin blockchain can be seen at https://www.statista.com/statistics/647523/worldwide-bitcoin-blockchain-size/

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 2.2. All of them were created before Bitcoin, and despite that, some of these attempts were only academic proposals, others were actually 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.[6]

So there is a question, what makes cryptocurrencies successful nowadays? It may be it's easy to use principle and no need for external hardware. Another critical component of cryptocurrencies discussed in this work is Blockchain. Simply, 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.[1]



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 [4]

2.1 Ethereum

2.2 Bitcoin

```
>You will not find a solution to political problems in cryptography.

Yes, but we can win a major battle in the arms race and gain a new
   territory of freedom for several years.

Governments are good at cutting off the heads of a centrally
   controlled networks like Napster, but pure P2P networks like
   Gnutella and Tor seem to be holding their own.

Satoshi
```

Figure 2.2: Satoshi Nakamoto at vistomail.com *Thu Nov 6 15:15:40 EST 2008* [1]

2.3 DigiByte

2.4 Decred

2.5 Monero

Chapter 3

Decentralization (Web3.0)

Chapter 4

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. In IPFS user needs only to know the hash of the requested file. He does not care about the location of the file.

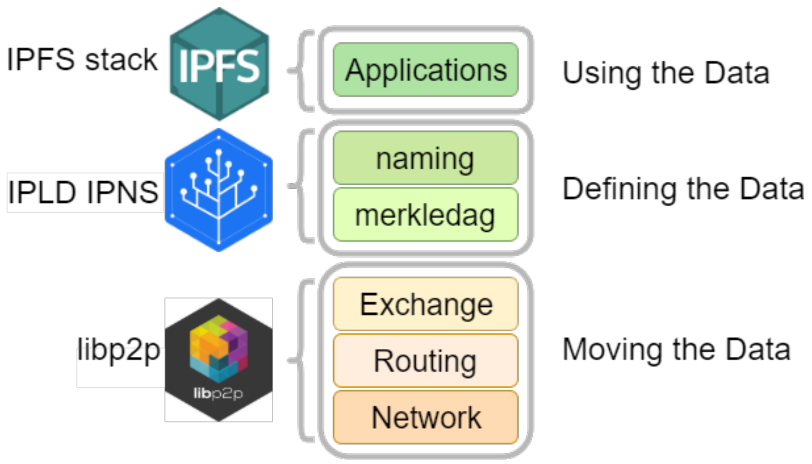


Figure 4.1: IPFS stack

4.0.1 Cluster

4.0.2 Node

4.0.3 CID

4.1 IPLD

4.1.1 Formats

4.1.2 Routing

4.1.3 Exchange

4.1.4 Objects

4.1.5 Files

4.1.6 Naming (IPNM)

Chapter 5

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.

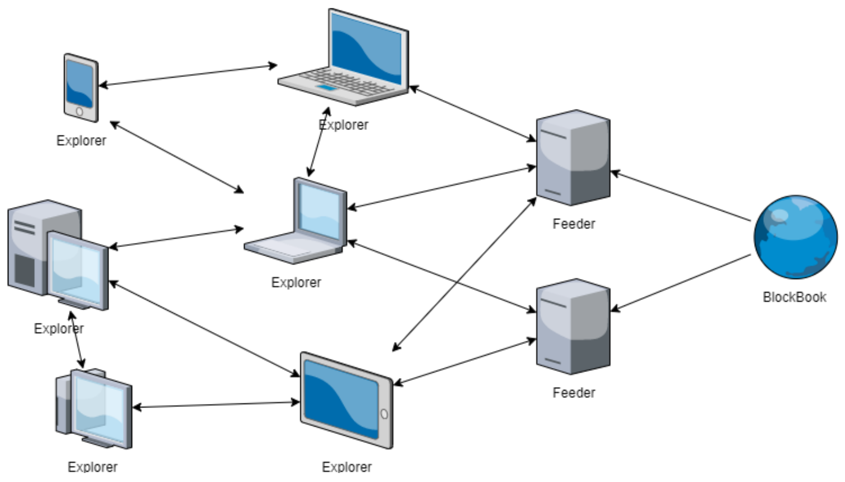


Figure 5.1: System design

5.0.1 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 community). For data storage Blockbook is using RocksDB[4] developed by Facebook which is NoSql database which stores only key-value pairs. Blockbook is providing fast api for acessing 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.

5.1 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 5.2. 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.

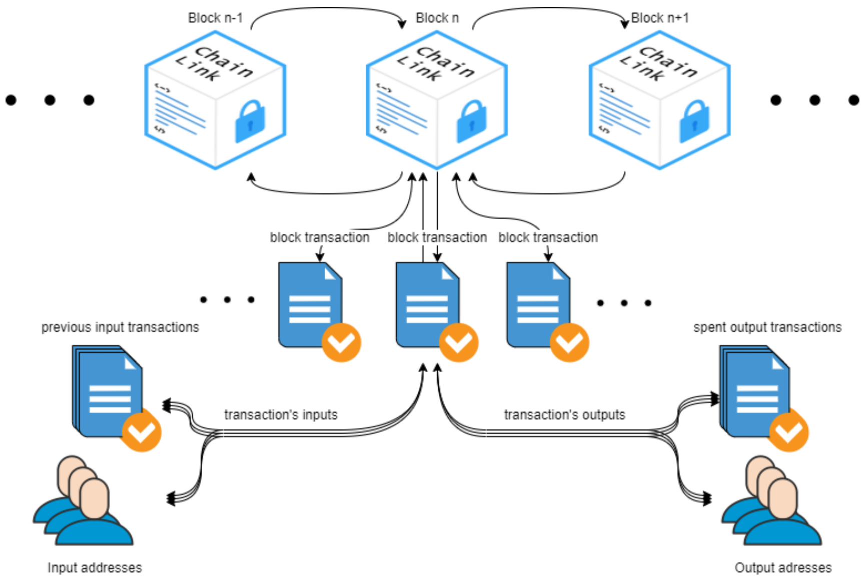


Figure 5.2: Feeder data storage structure

5.2 Explorer

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

[5]https://nodejs.org/

Chapter 6

Implementation

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

6.1 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.

6.1.1 Indexes

In my prototype, I tried 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 good solution for small user's databases, but is is still in alpha stage of developing, and it is not well optimized to store hundreds gigabytes of data. The biggest problem is that OrbitDB performs all queries locally. To preform 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 whole copy of database. This is not usable for our case, when we have database that has hundreds of gigabytes of data. [3]

- **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 high abstraction on top of the IPFS and provides simple API to securely store and index files. It uses *Cafe* peers to provides backups and indexing. Every data store is duplicated on several *Cafe* peers. When client is obtaining some data, it will contact one of the *Cafe* peers, and *Cafe* peer will resolve query for the client. This is a problem for

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

our solution, because using textile require lots of hardisk memory and does not solve problem with overloading *Cafe* peers. [5]

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

- **Dictionary** - 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 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).

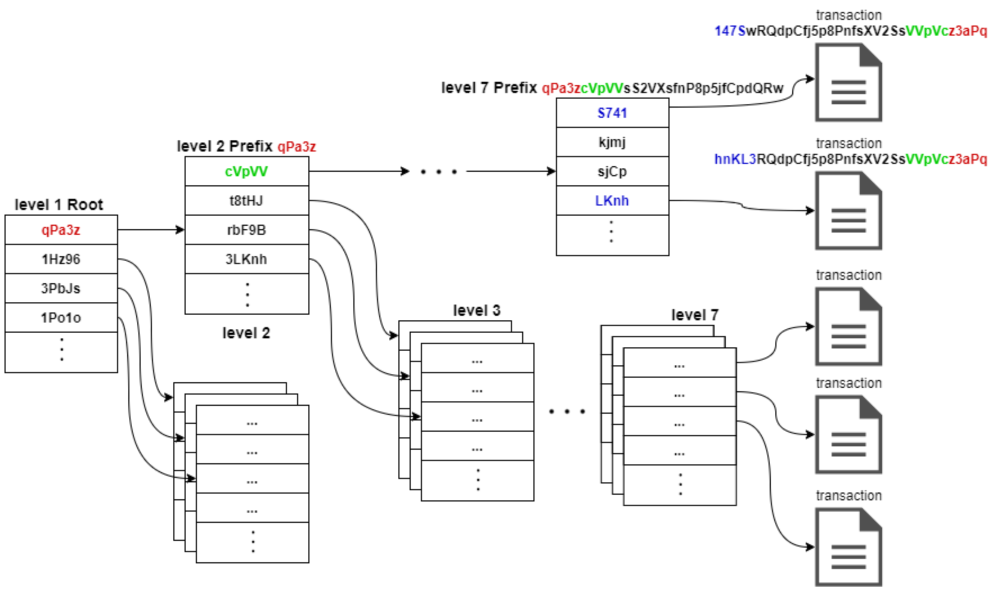


Figure 6.1: Reverse lookup by transaction hash

- **Reverse lookup** - this structure is inspired by Reverse DNS lookup[8] and it's principle can be seen on figure 6.1. For every item, that will be stored in this index, the key is reversed (for better selectivity) and split into the smaller substrings. First substring of every key is stored in root (level 1) dictionary and is pointing to another dictionary that consists of following substrings which have 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.

- **B+ Tree** - perhaps most powerfull index structure. With auto balanced B+ tree we can efficienty search objects by given key, and performe range selects. Example of

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

this structure can be seen on figure 6.2. By default there is limit of 32 items in one node (but can be change from 4 to 256).

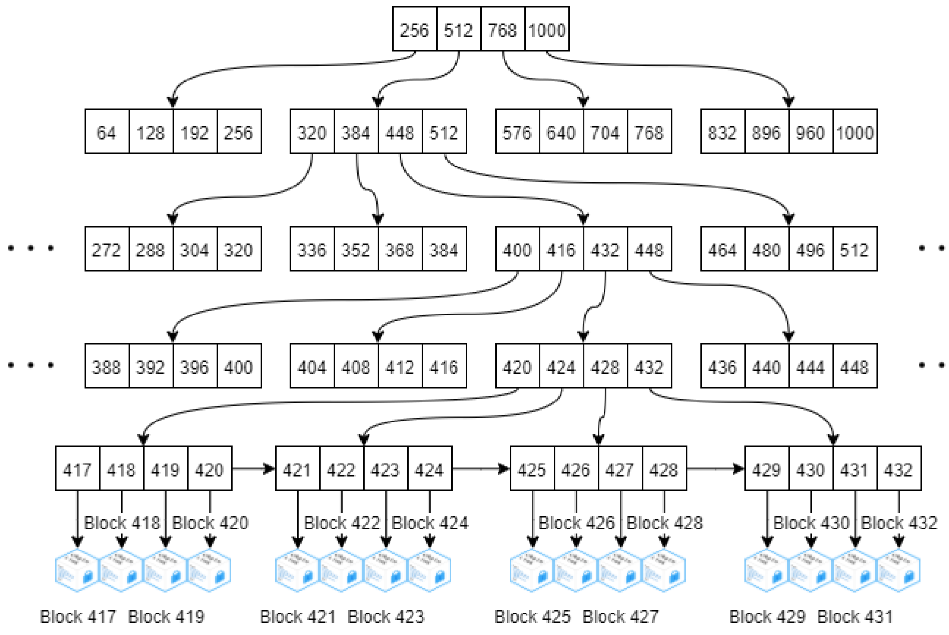


Figure 6.2: B+ tree index structure

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

6.2 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 on figure 6.3.

6.2.1 Running in browser as web aplication

Running in browser with GUI module provides simple user interface implemented as single page application. Browser's implementation of IndexedDB is used as a storage for IPFS as you can see at figure 6.4. Communication with other peers is provided though WebSockets. Every tab openned in browser is the same IPFS node. Openning new tab in Incognito mode or in different browser will spawn different IPFS node.

6.2.2 Running in node.js as REST api

Node.js[9] implementation of IPFS uses filesystem to store data (figure 6.5). On the top of ExplorerCore, there is simple web framework Express[10], where are routes (endpoints) defined. Currently supported routes are described in table 6.1. Every route supports query parameters `filter` uses for filtering results and `limit` which limits number of results.

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

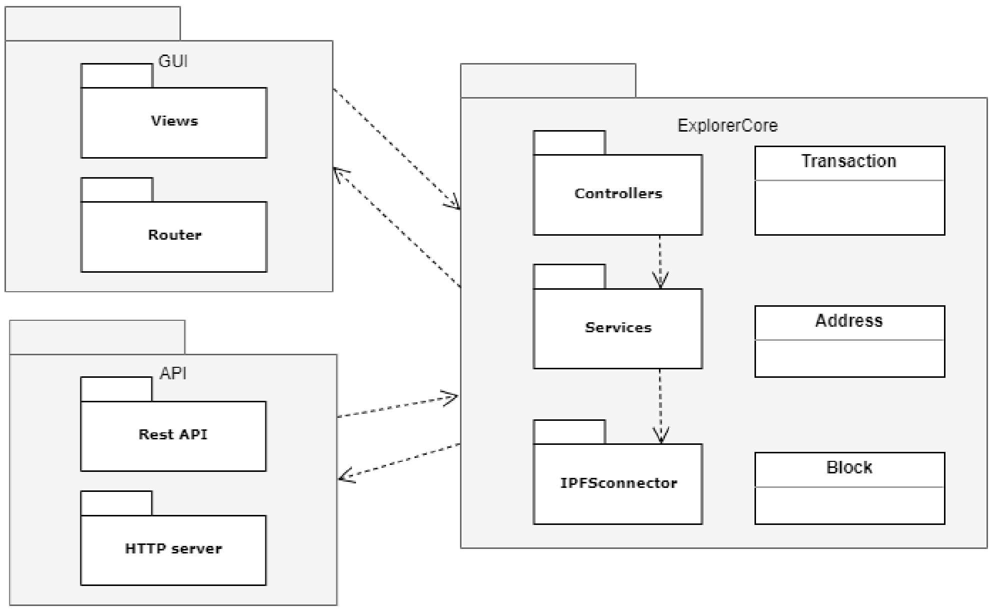


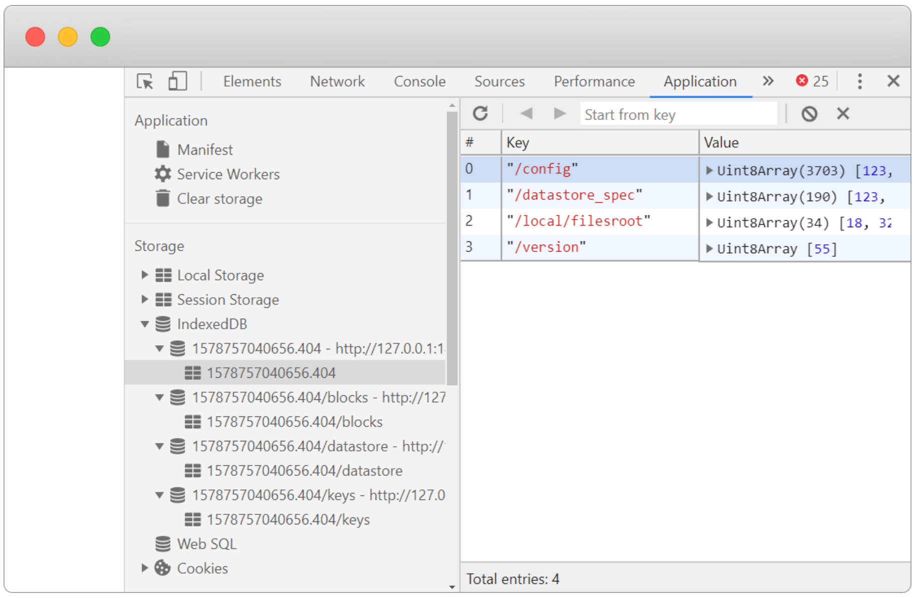
Figure 6.3: Explorer architecture

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 usefull for traversing objects in IPFS. If we want to get fifth transaction of 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 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

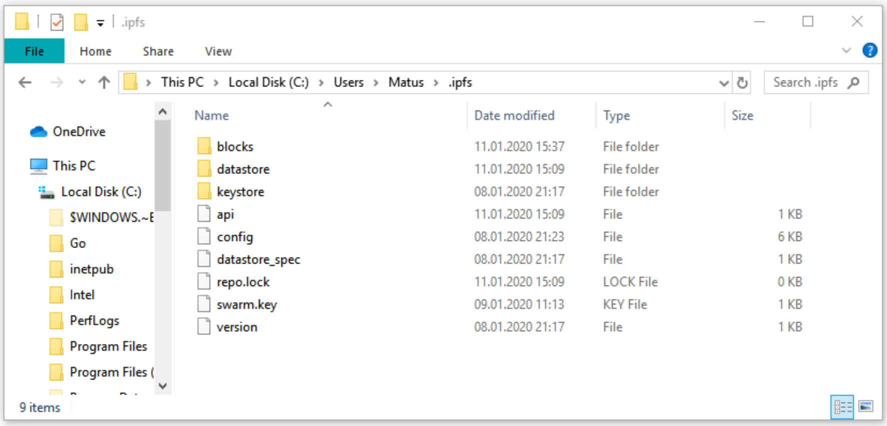
Table 6.1: Rest API explorer endpoints



#	Key	Value
0	"/config"	▸ Uint8Array(3703) [123,
1	"/datastore_spec"	▸ Uint8Array(190) [123,
2	"/local/filesroot"	▸ Uint8Array(34) [18, 32
3	"/version"	▸ Uint8Array [55]

Total entries: 4

Figure 6.4: IPFS storage in browser



Name	Date modified	Type	Size
blocks	11.01.2020 15:37	File folder	
datastore	11.01.2020 15:09	File folder	
keystore	08.01.2020 21:17	File folder	
api	11.01.2020 15:09	File	1 KB
config	08.01.2020 21:23	File	6 KB
datastore_spec	08.01.2020 21:17	File	1 KB
repo.lock	11.01.2020 15:09	LOCK File	0 KB
swarm.key	09.01.2020 11:13	KEY File	1 KB
version	08.01.2020 21:17	File	1 KB

Figure 6.5: IPFS storage in node.js

Chapter 7

Conclusion

Bibliography

[1] 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.

[2] 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.

[3] Mark Robert Henderson, S. P. *The OrbitDB Field Manual.* 2019. Available at: https://github.com/orbitdb/field-manual.

[4] Narayanan, A., Bonneau, J., Felten, E., Miller, A. and Goldfeder, S. *Bitcoin and cryptocurrency technologies: a comprehensive introduction.* Princeton University Press, 2016.

[5] 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/.

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