



BERN UNIVERSITY OF APPLIED SCIENCES

INFORMATICS SEMINAR

InterPlanetary File System
IPFS

Author:
Martin SCHMIDLI

Teacher:
Kai BRÜNNLER

Bern, April 30, 2017

Contents

Chapter 1

Abstract

IPFS (InterPlanetary Filesystem) is an opensource protocol which can be used to run a distributed filesystem. IPFS was invented to tackle the drawbacks of the Internet and the Internetprotocol suite we are using today.

To understand why IPFS was invented, we first have to understand the issues we have today. This report will outline how IPFS works, what kind of issues it tries to resolve and analyze the obstacles which have to be overcome to establish IPFS as an accepted and widely used technology.

Chapter 2

Basic Knowledge

This chapter will familiarize you with some technologies used in IPFS. It's important to understand those basics to fully understand how IPFS works.

2.1 Distributed Has Table (DHT)

Imagine a p2p network with many nodes in it. Data should be distributed evenly and the access to the data should be as efficient as possible. Every nodes

2.2 Merkle DAG

2.3 Bittorrent

2.4 Git

Chapter 3

Introduction

IPFS stands for InterPlanetary Filesystem. It's an opensource Internet Protocol which can be used to run a distributed filesystem. The developers didn't invent IPFS from scratch. In its core IPFS takes advantage of existing technologies like Kademlia DHT, BitTorrent and Git. The main intention behind IPFS is to tackle the issues we are having in today's Internet/Web. In the eyes of the inventors the protocol should be seen as an upgrade or even as a replacement of the existing technologies like HTTP. Global data distribution should be simplified and be implemented in the protocol itself rather than prescribe a separate distribution mechanism [?].

3.1 Origin

The Development of IPFS was started in 2014 by Juan Benet, former Stanford Student and founder of the company Protocol Labs . Protocol Labs and contributors of the Community are developing IPFS further [?].

3.2 Name

The name was chosen as a tribute to J. C. R. Licklider, a computer scientist who came up with an idea of a "intergalactic network" of computers in 1962 [?]. He imagined a global network of computers, able to talk to each other and exchange data. During his time working at DARPA (Defense Advanced Research Projects Agency) he influenced many people with his ideas. DARPA later started the ARPANET Project and laid the foundation stone for today's Internet. Many important technologies for example: TCP/IP were invented or funded during this project [?].

3.3 Project state

As of 19.04.2017 the Specifications of the IPFS Protocol are still being developed and hasn't completed yet. The developers state, that the core parts of the specs have reached a reliable or stable state. No official RFC request have been submitet. An Implmenetation of the Protocol, written in the programming language Go and some utilities have aready been published. Implementation in other programming languages Javascript and Phyton are in developing [?].

Different sources state different facts how the system will works. Many topics are still discussed intensively. This makes it very difficult to do a report about IPFS. The main data source of this report was the projects github page and a whitepaper release by Juan Benett.

Chapter 4

Today's Problems

The developers of IPFS state: The Internet/Web of today has many issues. The existing protocols we have today, have some major design issues or are not good enough anymore to satisfy the needs of today's Web and its Users [?].

4.1 Offline functionality

Imagine you are sitting with your colleagues at work. You all work together on a document. You are using a WebApplication to collaborate with each other. Suddenly the internet connection is lost. You are all sitting in the same network but you are unable to share your version of the document with the others. All the data needs to be synced with the backbone service for example Google Docs. and then down again to the other clients / your colleagues. The Applications should be able to talk to each other and shouldn't be dependant of a service somewhere in the internet.

4.2 Protocol

The most used Protocol from the Internet Protocol Suite by far is HTTP. It follows the Server-Client communication model. The Client establishes a connection to one server, sends requests and get responses back from the server. Even if there are load balancing mechanisms in place it will be served by the by one host only.

The alternative communication model would be Peer to Peer short. p2p. The Clients establishes a connection to multiple "peers" which will serve the requested data. Every peer will deliver a fraction of the data. Those data "pieces" will be downloaded simultaneously.

4.3 Permenancy

I guess every person which used an Internet Browser before has seen a message like "Error 404" or "Site not found".



Figure 4.1: Github 404 Error Page

The specific content you tried to reach has been deleted or has been moved by the sites provider. All the links provided are now useless. Every day content gets moved around or is deleted forever. Sometimes its just a silly picture of a cat but mostly its data somebody actually would have needed.

Nevertheless there are several companies already trying to solve this issue by archiving all the data available on the internet. The most well known would be "Internet Archive" with its "Wayback Machine" Webapplikation. They are using web crawlers to get the data and store them. The User is able to use theWayback Machine to view the webpage back then when the Internet Archive made a snapshot of it [?]. By October 2016 they managed to archive 273 billion webpages from over 361 million websites, which resulted in a data size of 15 Petabytes (15'000 Terabytes) [?].

4.4 Centralization & Proxy

Big data hosting companies like Google or Amazon own datacenters worldwide. Developer can buy the service from them to cluster their application data and distribute it all over the world. This ensures the availability of your Application even in case of a datacenter outage caused by a natural catastrophe or other technical issues. Those incidents aren't fictions and cost millions if they happen. Latest example, 28.02.2017 a site of the Amazon S3 System located in Northern Virginia got unavailable. Thousands of webpages and webapplications stopped working. A Amazon technician wanted to remove a small subset of servers. He entered the command wrong and more servers than planned were removed. This led to an 4h outage of the whole Site [?]. Estimated cost 150 Million US Dollar [?].

Clustering of data is not standard today. To make your data highly available is cost intensive. To achieve the same level of high availability cost effective without any kind of big service provider like Google is near impossible.

A second group of big data handlers are the so called CDNs. These Content Delivery Networks host data worldwide and act as a proxy for data requests. Services of these providers are used to bring content closer to the customers to reduce latency and download time.

4.5 Security

Man in the middle, armor the content not the tunnel

4.6 Privacy & Consumer Protection

In these days one of the main concerns of social media platforms are privacy issues and hate messages. Those companies, for example Facebook and Twitter, are facing huge fines in several countries because governments think they don't do enough to protect their users. [?]. Content shared with a service in the internet, mostly stays in the internet as it's rapidly shared and distributed. If it's out there it's out there forever. The content itself is hard to locate and remove. The data is addressed with links. The links or URL changes from hoster to hoster. If the affected user tries to notify the hoster of the data, for example a picture with the intention to damage the user's reputation, it might already be hosted by several other platforms.

Chapter 5

Security

5.1 Hash functions

Multihash

Multihash is a protocol invented and maintained by the company Protocol Labs. Multihash provides a Hash format ,which is used in the IPFS project to encode Hashes. Every hash generated will be stored with two addittional values; the Functioncode of the hash function, which was used to generate this hash and the length of the hash.

$$\text{Multihash Format} = \text{Hash Function Code} + \text{Hash Length} + \text{Hash}$$

The Hash functioncodes were set by the developers. For your own projects you can easily implement your own code hashfunction reference list.

codec	description	code
identity		0x00
sha1		0x11
sha2-256		0x12
sha2-512		0x13

Multihash should allow the software using IPFS to upgrade the hash functions more easily [?]. For example we hardcoded SHA1 as a hash function into our programm. If somebody would be able to break the SHA1 hash function we would have to replace the hash function. This would lead to longer hash values which might break our programm. By using a more generic format the hashfunction can be switched very quickly.

Example 5.1.0.1. sha1



Functioncode:

0x11, when we lookup the Multihash Function Table we can find **sha1**. This hash was generated by the sha1 hasfunction.

Length:

0x20 = 32, means $32 * 8 \text{ Bit} = 256\text{bit}$. The Hash is 256bit long.

Hash Digest: Hash Values

5.2 Encyption

Chapter 6

Architecture

6.1 Layers

IPFS contains of a stack of different software moduels.

Level	Layer	Purpose
1	Identitied	
2	network	
3	routing	
4	echange	
5	objects	
6	Files	
7	Naming	

6.2 Identities

All computers participating in an IPFS Network are called peers or nodes. During the initialization of a node, a 2048-bit RSA keypair (public, private) is generated. A hash function is used to generate the hash of the public key. This generated hash is then used as Peer ID [?].

$$\text{Peer ID} = \text{multihash}(\text{public key})$$

6.2.1 Trust

There is no central certification authority in place which can be used to check if another peer can be trusted. The whole system has been designed to be self-verifying.

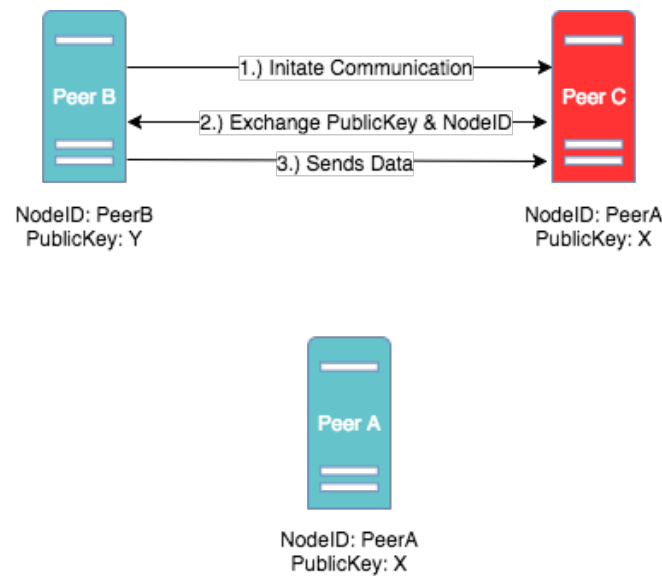
Example 6.2.1.1. Communication.

Peer A contacts another peer B. The Nodes exchange Node ID and Public Key. Peer A generates the hash of the public key of peer B. If this generated hash doesn't match the NodeID of peer B the connection will be terminated.



Example 6.2.1.2. Attack scenario

Peer C tries to steal the identity of peer A. Peer C is using the public key and the Node ID from peer A. Peer B will establish a connection with peer C as the public key is valid and the Node ID matches. The sent data from B to C will be encrypted by Peer A's public key. As Peer C doesn't hold the private key, the data can't be decrypted and therefore it's useless.



6.3 Data Addressing

Compared to HTTP where data is addressed by URL's is IPFS the data is addressed by its contents hash.

To understand how the data addressing in IPFS works, first a little example how its done with HTTP.

6.3.1 HTTP

Data which is access by HTTP is adressd with its hostname, port, path and the filename or searchword [?].

Example 6.3.1.1. URL schema used for HTTP

```
http : // hostname [ : port ] / path [ ? searchwords ]
https://raw.githubusercontent.com/github/gitignore/master/TeX.gitignore
```

If the location of the file changes and no redirection was created, the link gets useless.

6.3.2 IPFS

IPFS goes a different way. Data added to the global IPFS can be addressed by the hash of its content. When the content of the file changes so does the hash value. The changed content will be added to the Network. The older content will still be accessible with the old hash value. This garanties the persistance and the versioning of the data.

Example 6.3.2.1. Add a file

We create a demo file

```
GNU nano 2.0.6 File: foo.txt
foo
```

We add the file foo.txt to the IPFS Network. The content gets hashed and creates an key to this specific data. **QmYNmQKp6SuaVrpgWRsPTgCQCnpXUYGq76YEKBXuj2N4H6** is the key to our file with the content "foo".

```
[N-MACBOOK-3333:Desktop martinschmidli$ ipfs add foo.txt
added QmYNmQKp6SuaVrpgWRsPTgCQCnpXUYGq76YEKBXuj2N4H6 foo.txt
```

This hash value can now be used to access the content from any IPFS client. Example with cat

```
[N-MACBOOK-3333:Desktop martinschmidli$ ipfs cat QmYNmQKp6SuaVrpgWRsPTgCQCnpXUYGq76YEKBXuj2N4H6
foo
```

Let's change the content of "foo.txt".

```
[N-MACBOOK-3333:Desktop martinschmidli$ echo "new Foo" >> foo.txt
[N-MACBOOK-3333:Desktop martinschmidli$ cat foo.txt
foo
new Foo
```

Add the file again. Because we changed the content, the hash value will be different this time. New Key: **QmbYzb3nScopAnfkoUpRWUFVv856uSWpSRc2KYM1FSBJxr**

```
[N-MACBOOK-3333:Desktop martinschmidli$ ipfs add foo.txt
added QmbYzb3nScopAnfkoUpRWUFVv856uSWpSRc2KYM1FSBJxr foo.txt
```

The system is actively deduplicating data. If existing contents gets added for example two times the same picture they will get the same hash value, but the raw data will be stored just once in the IPFS network.

Example 6.3.2.2. Deduplicating

We add two files, same content but different names **foo2.txt** and **foo3.txt** with the content "new Foo"

```
[N-MACBOOK-3333:Desktop martinschmidli$ echo "new Foo" >> foo2.txt
[N-MACBOOK-3333:Desktop martinschmidli$ echo "new Foo" >> foo3.txt
[N-MACBOOK-3333:Desktop martinschmidli$ ipfs add foo2.txt
added QmYxEG2M5LhqPwoRXhvhTWPVKtapZNXMHtL1TgPypgqNDK foo2.txt
[N-MACBOOK-3333:Desktop martinschmidli$ ipfs add foo3.txt
added QmYxEG2M5LhqPwoRXhvhTWPVKtapZNXMHtL1TgPypgqNDK foo3.txt
```

Both times the generated hash is

QmYxEG2M5LhqPwoRXhvhTWPVKtapZNXMHtL1TgPypgqNDK

6.4 Mutability - IPNS

All the data added to the IPFS Network is immutable. The data can't be changed anymore. New content generates a new data and therefore a new hash key. If you change your data, people with the old hash key (link) will always be directed to the old version of the data. To solve this issue, the developers created the InterPlanetary Naming System (IPNS). IPNS creates a permanent link using the ID of the node which is publishing the data.

Example 6.4.0.1. IPNS

We publish new data to the network and save the hash value of the content

```
N-MACBOOK-3333:~ martinschmidli$ echo "test Schmidli" | ipfs add
added Qmd7J7FAWkXAGarT3jXZ9rT7JPzzn7YKeyRok6vv6BwLMd Qmd7J7FAWkXAGarT3jXZ9rT7JPzzn7YKeyRok6vv6BwLMd
```

Next we publish a new IPNS record using the hash of the data

```
N-MACBOOK-3333:~ martinschmidli$ ipfs name publish Qmd7J7FAWkXAGarT3jXZ9rT7JPzzn7YKeyRok6vv6BwLMd
Published to QmYVWNTT3QEa3aiLUa9eKEQR9x6RHxazMcGf2BVkSe8XjV: /ipfs/Qmd7J7FAWkXAGarT3jXZ9rT7JPzzn7YKeyRok6vv6BwLMd
```

/ipfs/Qmd7J7FAWkXAGarT3jXZ9rT7JPzzn7YKeyRok6vv6BwLMd is the published content. The content link has been linked to our Peer ID.

If we resolve the Peer ID, we get the link to the published content. In this case it is listing the ipfs link to our test data

```
N-MACBOOK-3333:~ martinschmidli$ ipfs name resolve QmYVWNTT3QEa3aiLUa9eKEQR9x6RHxazMcGf2BVkSe8XjV
/ipfs/Qmd7J7FAWkXAGarT3jXZ9rT7JPzzn7YKeyRok6vv6BwLMd
```

Let's assume we want to update our data but we want to keep the same link to it.

```
N-MACBOOK-3333:~ martinschmidli$ echo 'test Schmidli NEW DATA' | ipfs add
added QmZrMGGzWU7uBskBwBZyMyBPKNqtSHnupr78JLxmqwbF3 QmZrMGGzWU7uBskBwBZyMyBPKNqtSHnupr78JLxmqwbF3
N-MACBOOK-3333:~ martinschmidli$ ipfs name publish QmZrMGGzWU7uBskBwBZyMyBPKNqtSHnupr78JLxmqwbF3
Published to QmYVWNTT3QEa3aiLUa9eKEQR9x6RHxazMcGf2BVkSe8XjV: /ipfs/QmZrMGGzWU7uBskBwBZyMyBPKNqtSHnupr78JLxmqwbF3
```

The new data has been published and can now be accessed by the peer ID. The old data has been removed as a reference and is no longer accessible by the Peer ID.

```
N-MACBOOK-3333:~ martinschmidli$ ipfs name resolve QmYVWNTT3QEa3aiLUa9eKEQR9x6RHxazMcGf2BVkSe8XjV
/ipfs/QmZrMGGzWU7uBskBwBZyMyBPKNqtSHnupr78JLxmqwbF3
```

6.4.1 Restrictions

If we publish new data with IPNS, the ipfs path of the new data will be linked with our Peer ID. That means at the moment it's only possible to have one static link per peer, which we can use to link changing content.

6.5 Exchange - Bitwsap

Bitswap is a bittorrent-inspired software module used to exchange data blocks between the nodes of the IPFS network. When a user requests a piece of data the specific blocks are added to the nodes "want_list" list. Bitswap contacts other connected peers and sent them the "want_list". Peers keep track of the blocks which they own by putting them into their "have_list". If the reciever has blocks which another node requested, he will send the blocks back to the requester. 'When the blocks arrive, the requester will imidiately send out a "Cancel' signal. Other nodes will stop sending those blocks. After the blocks arrived the requester move those blocks from the want_list to the have_list [?].

6.5.1 Data

Small Files

Big Files

A ipfs object is not necessaraly a hole file. A file can be split up in smaller blocks. If we add a file bigger than 256kB, the file will be split up into blocks. The generated hash will be the root node in the DAG. This root node will contain the links to the subblocks. If a node requests a block, this block can come from any node owning one of these blocks even if its not the same "file" [?]. Mutlible IPFS objects can share blocks.

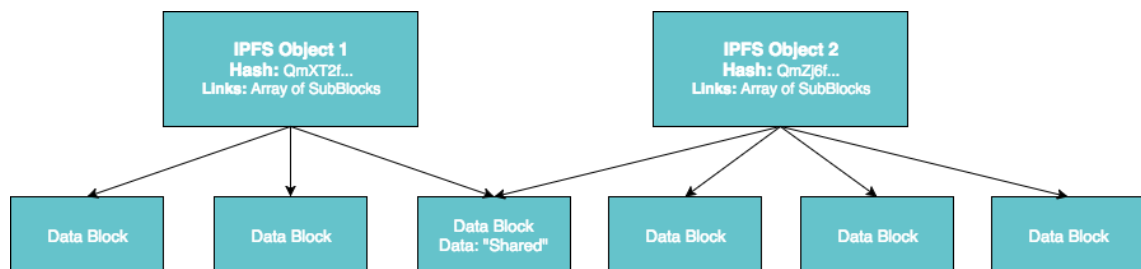
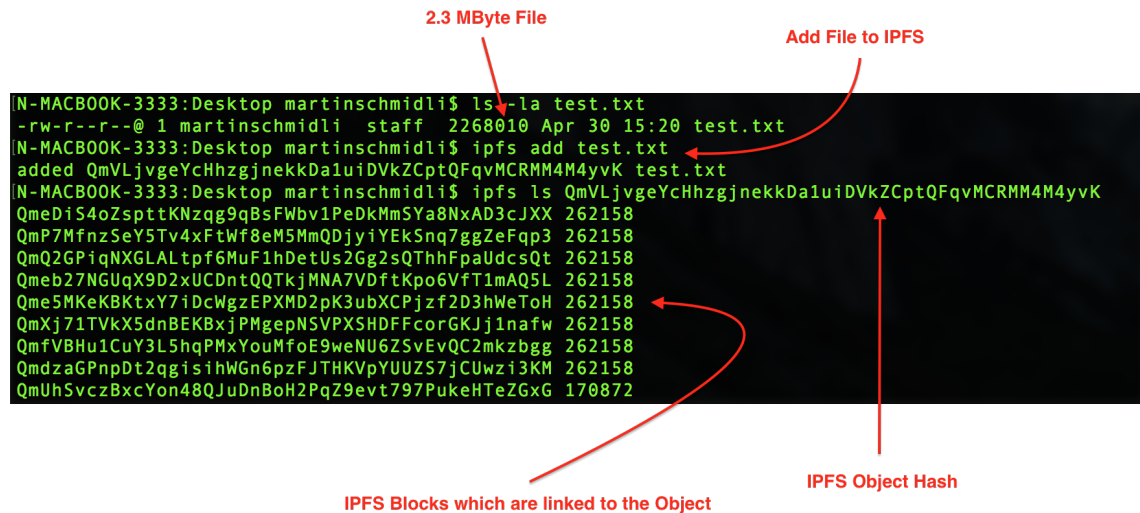


Figure 6.1: Abstract visualization how blocks are handeled

Example 6.5.1.1. Add big File



The image shows a terminal window with the following commands and output:

```
N-MACB00K-3333:Desktop martinschmidli$ ls -la test.txt
-rw-r--r--@ 1 martinschmidli  staff  2268010 Apr 30 15:20 test.txt
N-MACB00K-3333:Desktop martinschmidli$ ipfs add test.txt
added QmVLjvgeYcHhZgjnekDa1uiDVkZCptQFqvMCRMM4M4yvK test.txt
N-MACB00K-3333:Desktop martinschmidli$ ipfs ls QmVLjvgeYcHhZgjnekDa1uiDVkZCptQFqvMCRMM4M4yvK
QmeDiS4oZspttKNzqg9qBsFWbv1PeDkMmSYa8NxAD3cJXX 262158
QmP7MfnzSeY5Tv4xFtWf8eM5MmQDjyiYEKSnq7ggZeFqp3 262158
QmQ2GPiQNXGLALtpf6MuF1hDetUs2Gg2sQThhFpaUdcsQt 262158
Qmeb27NGUqX9D2xUCDntQQTkjMNA7VDftKpo6VfT1mAQ5L 262158
Qme5MKeKBKtxY7iDcWgzEPXMD2pK3ubXCPjzf2D3hWeToH 262158
QmXj71TVkX5dnBEKBxjPMgepNSVPXSHDFFcorGKJj1nafw 262158
QmfVBHu1CuY3L5hqPMxYouMfoE9weNU6ZSvEvQC2mkzbgg 262158
QmdzaGPnpDt2qgisihWGn6pzFJTHKvpYUUS7jCUwzi3KM 262158
QmUhSvzcBxcYon48QJuDnBoH2PqZ9evt797PukeHTeZGxG 170872
```

Red annotations point to specific parts of the output:

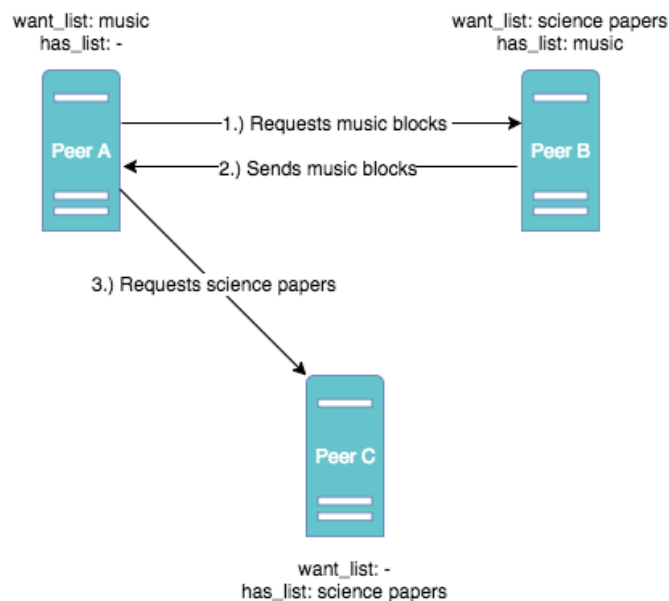
- 2.3 MByte File**: Points to the file size '2268010' in the first line.
- Add File to IPFS**: Points to the command 'ipfs add test.txt'.
- IPFS Object Hash**: Points to the hash 'QmVLjvgeYcHhZgjnekDa1uiDVkZCptQFqvMCRMM4M4yvK'.
- IPFS Blocks which are linked to the Object**: Points to the list of blocks in the 'ipfs ls' output.

Figure 6.2: IPFS Objects Example

6.5.2 Exchange Strategies

When a bitswap node requests data (Peer A), the other nodes (Peer B) will only send the requested data if they receive blocks in return. Most of the time Nodes store very different data and the amount of blocks which they could offer in return is very limited. Sometimes Node don't own any data the other node is needing. This puts the requesting node (Node A) in a bad position as it has nothing he could offer in exchange. To resolve the situation, Node A will start to download the data Node B has on its want_list after all his own data requestes are completed. The own needs have a higher priority.

Example 6.5.2.1. Node A wants music and owns no data. Node B wants blocks of science papers and owns music blocks → Node A has no blocks which Node B actually needs.



After Peer A got all the music blocks he wanted, he will start to look for science paper blocks.

Automatical Download

This exchange mechanism leads to data download even if the node doesn't actually need the data by itself. The data are cached and increase the availability of the blocks. This strategy is not yet implemented in bitswamp. Today's implementation is syncing blocks without any conditions and doesn't fulfill the defined specifications.

There are ongoing discussions if this feature should be enabled by default or if there should be a config option to enable it [?]. If this feature is enabled by default, IPFS nodes would download data without any knowledge of the user. This would lead to an extensive usage of the local storage and could lead to legal problems. To avoid the legal problematic, there needs to be a system in place to avoid the download of Copyright Protected Data and to implement Digital Right Management (DRM). If and how this will be implemented is currently discussed in the community [?].

6.5.3 Data Availabiliy

6.5.4 Difference to Bittorrent

Bittorrent nodes can only request blocks from one torrent.

6.6 Routing

6.7 Network

Chapter 7

Today's Use Cases

7.1 Mediachain Labs

Chapter 8

Outlook

Encryption built in The Question is: "Is there a future for IPFS"?

Chapter 9

Conclusion

Bibliography

- [1] <https://github.com/ipfs/specs>, 29.04.2017
- [2] <https://github.com/ipfs/ipfs>, 24.07.2017
- [3] <https://www.linkedin.com/in/jbenetcs/>, 28.04.2017
- [4] <https://www.youtube.com/watch?v=HUVmypoX9HGI> 4:30, 22.04.2017
- [5] <http://www.internetsociety.org/internet/what-internet/history-internet/brief-history-internet>, 22.04.2017
- [6] <https://ipfs.io/>, Sector: The web of tomorrow needs IPFS today, 22.04.2017
- [7] https://en.wikipedia.org/wiki/Internet_Archive, 22.04.2017
- [8] <https://blog.archive.org/2016/10/23/defining-web-pages-web-sites-and-web-captures>, 22.04.2017
- [9] <https://aws.amazon.com/message/41926/>, 23.04.2017
- [10] <http://www.npr.org/sections/thetwo-way/2017/03/03/518322734/amazon-and-the-150-million-typo>, 23.04.2017
- [11] https://www.nytimes.com/2017/03/14/technology/germany-hate-speech-facebook-tech.html?_r=0, 23.04.2017
- [12] <https://www.w3.org/Addressing/HTTPAddressing.html>, 23.04.2017
- [13] <https://github.com/ipfs/faq/issues/238>, 24.04.2017
- [14] <https://github.com/jbenet/random-ideas/issues/1>, 24.04.2017
- [15] <https://github.com/ipfs/specs/tree/master/bitswap>, 29.04.2017
- [16] <https://github.com/ipfs/examples/tree/master/examples/data>, 29.04.2017
- [17] <https://github.com/ipfs/faq/issues/47>, Q: but bitswap says..., 29.04.2017
- [18] https://www.reddit.com/r/ipfs/comments/3m351b/discussion_permanent_content_dmca_and_illegal/?st=j24nd4on&sh=86fd55ce

List of Figures