Using Blockchain Technology in Distributed Storage Systems

Bruno Produit
Institute of Computer Science
University of Tartu
produit@ut.ee

June 10, 2018

Abstract

This report is an analysis of the distributed file storage systems Storj and IPFS. More specifically this report presents the distributed storage mechanism linked with blockchain technology. Both systems are analyzed and compared.

Contents

1 Introduction

As storage technologies evolve, many new schemes to store data are invented. Companies like Dropbox, Drive, iCloud and other cloud services give the user the possibility to scale his storage easily and pay for it. New ideas are emerging for distributed storage and this report talks about those. The main idea behind this is that many people have some free storage space on their computer drive and it could be useful to rent it out. More recently blockchain technology has enabled easy incentivization for those distributed storage schemes. Two main competitors, Storj [WBBB14] and "Interplanetary file system" (IPFS) [Ben14] are going to be analyzed in this report.

2 Storj

2.1 Overview

Storj is a distributed storage system, which enables users to sell and buy storage space. The idea is to share the remaining free disk space and sell it for a market-given price. This storage can then be bought to be used by anybody. This "metadisk" is backed by a blockchain for exchange and incentivization of the proof of retrievability. Storj can be compared to Dropbox. The idea is to pay for a certain amount of storage, which then can be used to store the users own data. Storj proposes a client program, which can be used by users on their devices.

2.2 Storage

Globally, Storj works as a middleware which connects the blockchain storage contract with the actual storage device. Figure 1 shows where the Storj client is situated in the storage stack.

Figure 1: Storj protocol

In order to store the actual files, the Storj client implements multiple techniques for redundancy, privacy and other purposes. The main techniques are explained below.

2.2.1 Sharding

Sharding is the name of the process which Storj uses to cut the data into pieces. First, the client encrypts the data and then cuts it into pieces, which are distributed in the network independently. This is used for purposes such as keeping a maximum size of data pieces, redundancy and balanced distribution of data in the network.

From a client point of view the system can be viewed as in Figure 2:

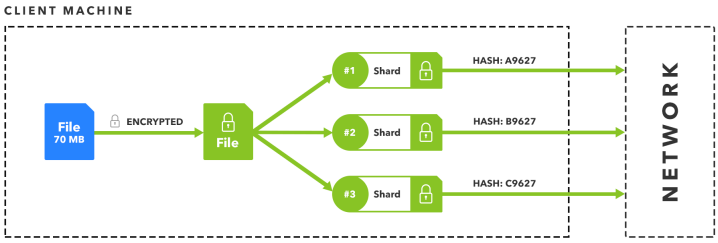


Figure 2: Client View of Storj [WBBB14]

2.2.2 User audit

One interesting aspect of Storj is the audit mechanism. Storj gives the possibility to the user and any peer in the network to check that the contract has been honored from both sides with this mechanism. The idea is to publicly check that payment has been wired, by checking the status on the blockchain and check that the transaction has been incorporated and signed in the next block. This mechanism is not made by Storj but is a standard component of Blockchain. The real mechanism of audit itself is not about payment checking but storage checking, which is the other side of the contract. As such every peer in the network is also able to check that the data has been stored with the audit mechanism. This mechanism works in the following way: Each "shard" (block of bytes from a file) has its own hash and is stored separately. Each of these shards is incorporated in a Merkle tree with the other shards on the network. The previously explained system is then implemented. The Merkle tree root is incorporated in the blockchain. One more problem remains though. It is possible to construct this tree just by knowing the hash and without storing the file itself. The solution implemented for that in Storj is the proof of retrievability. The following audit mechanism is implemented: every peer can be a verifier, who reads the root of the Merkle tree from the current block. The verifier asks the path in the tree to the storer and asks for a proof of retrievability.

2.2.3 Proof-of-Retrievability

The proof of retrievability is an interactive mechanism to prove the existence of a fresh copy of a shard on the storer side. The protocol gives the verifier the proof that a shard exists on the disk of the storer at a certain timestamp. It can thus not be considered a proof of retrievability, but more a proof of existence of a fresh shard. The system works as follows: The verifier sends a challenge

to the storer. This challenge is a list of salt values. For each salt, the storer concatenates it with the shard which has to be proven and hashes them together. These hashes are hashed together in a Merkle tree, from which the root is sent to the verifier. In this case the storer must have the shard as he has to hash it with the salts. Thus it proves the existence of the shard. The freshness is given by the fact that the verification can be asked at any time and by anyone. The system can be seen below.

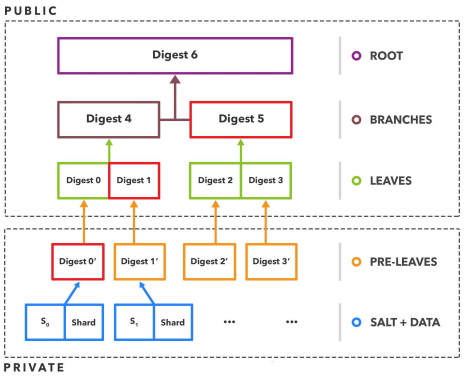


Figure 3: Storj Audit mechanism [WBBB14]

Storj gives also the possibility to verify with a partial audit, where the salts (random numbers provided by verifier) are concatenated to blocks of bytes from the shard. Each block of bytes is an equal length block of the shard (except the last block). This gives the storer the possibility to have a more lightweight audit.

2.2.4 Kademlia extended

When it comes to location of storage of the shards themselves, Storj uses a distribution system based on distributed hash tables implemented with a network protocol called Kademlia [MM02].

The extended the protocol includes more possibilities than just store/lookup/retrieve. These extensions are messages requesting and responding audit requests. Kademlia is a protocol which implements a distributed hash table. The protocol gives three types of messages to the peers. Those 3 message types are used to lookup a value, retrieve a value or store/delete a value. The system works as a binary tree, which is used to decide where the values have to be stored. As it is a hash table, the basic storage is a key-value store. This key-value store is distributed according to the routing tree.

The routing binary tree is constructed with a specific mechanism: Each peer picks a 160-bit random address. The distance to every other node is the xor of the addresses. For each different bit a branching is done in the binary tree. The distance weight is the integer value of the difference bits. The values are then stored by keys, which are the addresses. If a file has a certain hash, it will be stored at the nearest neighbor down the tree. The xor operation makes sure that every peer will have the same routing tree. This is due to the fact that $a \oplus b == b \oplus a$ and associativity. The routing table system can be seen as it is below.

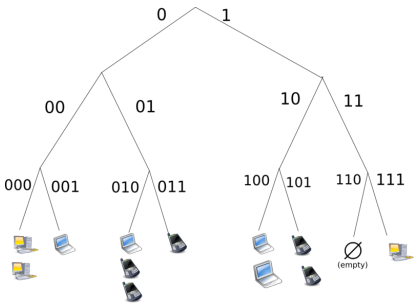


Figure 4: Kademlia routing table [Wik18b]

2.2.5 Redundancy

Storj shards are the containers for data in this system. Those shards are distributed throughout the network following the Kademlia protocol. As in a distributed system, peers connect and disconnect all the time, it is important to ensure redundancy of the data, in order to be able to retrieve it at all times. The redundancy scheme of Storj is only described as a paragraph in the white paper [WBBB14] and there is no further information available. It is said that the redundancy is made on the shard level, where peers should be mirrored. As a second solution, Storj proposes that in the next versions, K-of-M Reed-Solomon erasure coding should be used. This would be done on the file level and not the shard level. As such the coding would create M parity shards for K data shards. Those shards would then simply be stored on the network as any other shards. The redundancy scheme here is only at its first iteration and thus not developed further.

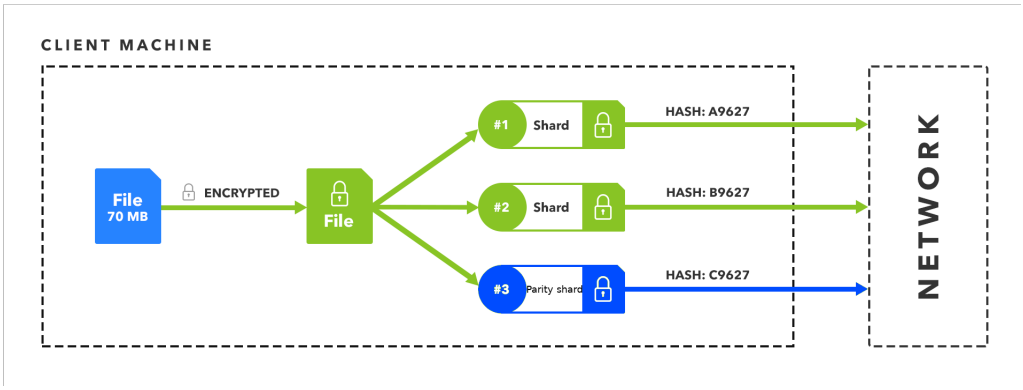


Figure 5: Redundancy to be implemented in Storj [WBBB14]

2.3 Cryptography

The crypto primitives used in Storj are not particularly standard in the context of distributed storage. The files themselves are encrypted with *AES256-CTR*, where CTR is the counter mode. The addresses themselves are hashes made of *ripemd-160(sha-256())* in order to have a 160-

bit hash for the Kademlia system. It is also not explained in the whitepaper why *sha-256* is used before ripemd, other than the fact that it is a valid bitcoin address, which can be used for payment. It is also not very standard practice to see ripemd160, which it is also not explained in the whitepaper.

2.4 Blockchain

Storj says it is "payment agnostic", meaning that the payment and storage contract can be implemented on any platform. Storj is just connected to blockchain, but does not implement this part of the system. Storj uses the blockchain for two use-cases: metadata and renting out space.

3 IPFS

3.1 Overview

IPFS [Ben14] is a P2P file system, which takes a broader approach to storage than Storj. The idea is to create an addressable network, which can be private or public (private content is encrypted). The idea is to recreate a decentralized web-like platform. IPFS could be compared to a normal file system, which is accessible to everyone. The main implementation is written in the Go programming language [Goo18]. Other implementations in JavaScript and python exist.

3.2 Storage

The storage of IPFS is cut in multiple layers. Like the OSI layering, IPFS layers implement a specific function and are defined to be flexible and compatible with other systems. In this regard, IPFS is a mature protocol which permits to be easily bridged to other services.

3.2.1 Identities

Identity is the layer which takes care of the name of the node. In Storj the addresses are the hash values of the public key of a peer, but in IPFS it is more complicated. The addresses are calculated as a S/Kademlia (extension of Kademlia) puzzle, which gives two advantages to the system. First it pushes peers to keep their address, and secondly, it hinders Sybil attacks [Wik18a] on the network (creating more than 50% of the nodes on network), as creating a new address is expensive. The Kademlia puzzle is produced by generating a key pair, hashing it iteratively until it is preceded by a chosen number of zeros (difficulty parameter). These final hash then corresponds to the node-ID.

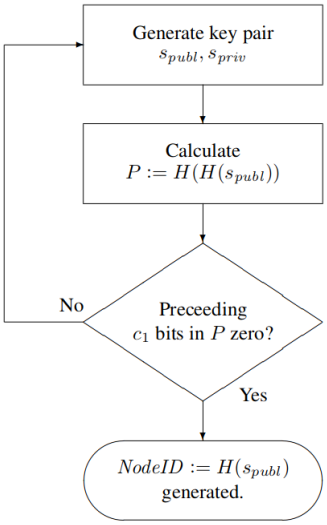


Figure 6: static Kademlia Puzzle[BM07]

3.2.2 Network

The network layer is comparable to the OSI Transport layer, as it transports the information in a P2P mode and ensure integrity, connectivity, reliability and authenticity of the data. It is made flexible so that any overlay protocol can be used for the transport itself. It also implements NAT traversal in order to be able to communicate to private IP behind NAT. This layer is a big layer, with many interesting characteristics from the networking point of view. We omit a detailed description of this layer as the subject of this report is the cryptographic side of the protocols. That said, the network layer is based upon UDP and implements most of standard components over a UDP protocol. It follows a routing table and is peer to peer.

3.2.3 Routing

The routing layer is used to construct the routing table. In order to do this, a distributed hash table is used (Kademlia). As Storj, IPFS uses Kademlia, thus the routing layer is very comparable to the routing happening in Storj. Kademlia is used in IPFS in its extended form.

3.2.4 Exchange

The exchange layer is the actual data layer, where nodes exchanges blocs. The protocol used is called the bitswap protocol. The bitswap protocol is a Bittorrent swarm (micronetwork), which is not limited to only 1 file. The idea of bitswap is that, because peers are to exchanging a lot of data, either for proofs or for redundancy and storage, each peer sets up lists of hashes which he would like to get and lists of hashes he is able to give away. These lists are exchanged in the network, so that the routing and exchange of actual data is facilitated. This also gives the possibility to nodes to solve bittorrents seeding (re-uploading the acquired data) issue, as

those bitswap lists could be bound to an incentive in order to push peers to exchange data. This mechanism is implemented in Filecoin (see section 3.4). With this exchange market, every file which is gathered (if a node visits a website for instance), is also present on the visiting node. This visiting node is also going to be able to seed that data, like in Bittorent. Thus objects are even more redundant when more people ask for them, which hinders distributed denial of service (DDOS). If more people access a website, the website becomes more redundant, as opposed to the standard server-client model.

3.2.5 Objects

The object layer is the layer which takes care of blocks, and addresses those in order to be able to reconstruct data. The object layer is built on a git data structure, which is a Merkle Directed acyclic graph (DAG). Furthermore, objects can be encrypted (see section 3.3). The Merkle DAG is a data structure which extends the Merkle tree. It is a tree which can contain symbolic links to other leafs, if no cycle is induced with it. As IPFS tends to be like the web, so that there should be the possibility of links and hierarchy (as a website). This is solved by the fact that objects can be linked by being a predecessor in the Merkle DAG. This gives the possibility to files to be interleaved, which is not possible with a simple tree. Paths and recurrent parenting are also possible (see section 3.2.7).

3.2.6 Files

Files layer is the uppermost layer which permits the user to store files like one would store in a regular file system. Multiple implementations of this layer work as a filesystem on Unix, by using system calls, Unix addressing and fuse. A file is made of a list of hashes (called "objects") of the Object layer and meta-data regarding the file. The files themselves are defined as a structure which contains a list of objects. As those objects can be lists, the file size is not restricted and can be seen as a balanced tree of objects. This methods permits to have a file structure which resembles Linux inodes (standard file type in the Linux kernel), but without size restrictions. The contained meta-data for files is the same structure as git (versioning system), with commits for versioning. As the files are not encrypted (encryption is at the object layer), it is simple to change a part of a file or update the meta-data, as only one object will be changed and a new commit with the new hash for the object can be inserted.

3.2.7 Naming

IPFS is also made of a naming system which is self certifying. That means that each link can be certified to be the good destination As the locations are the node-id's, it can be verified that they correspond to the public key which signs the files. The problem in IPFS is that each object has its own link and is not mutable, as even a single bitswap will change the hash completely. Thus IPFS created the IPNS naming system, which gives the possibility to put locations on files which are mutable. This naming system is also compatible with Domain Name System (DNS), by including the non mutable link in the TXT record (comment section in DNS). This allows for bridging between DNS and IPNS as well as other systems such as Namecoin.

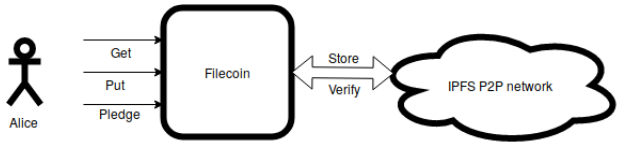
The mutable links work by having a link to the node-id of the target and then the corresponding file hierarchy of the given sub-DAG. In reality it looks like, for example, */ipns/XLF2ipQ4jD3UdeX5xp1KBgeHRhemUtaA8Vm/docs/ipfs*

3.3 Cryptography

The IPFS file system uses "multihash" which is just a data structure with a header and hash, where the header specifies the hash function used. Thus no precise hash function is specified. The object structure is extended to be wrapped in an encrypted object and a signed object. These two objects are defined to be a structure where the bytes are followed by the hash and the signature. Interestingly IPFS does not only use cryptography for the sake of encryption but for the routing, the addresses and the file location. As for the encryption of files, on the contrary to Storj it is not mandatory to encrypt files, as they can also be a website for instance. If someone wishes to have files for himself, he can do it by encrypting them himself. The encryption is done on the object layer, which defines a structure like multihash, where the specificity of encryption are given as metadata and the data is then encrypted using those options. It is up to the implementation to use good cryptographic primitives.

3.4 Filecoin

In order to incentivize IPFS, there exists a layer on top, called Filecoin, which creates a decentralized market for storage. As IPFS is opt-in for storage, the Filecoin blockchain gives the incentive to sell and buy storage on IPFS. It is important to note that the presented schemes below are made non-interactive, in order for everyone to be able to mine. Filecoin can be seen as a supplementary layer on top of IPFS.



First there are two separate work-flows which are two different market called storage market and retrieval market.

The work-flow for storing data is the following, in two phases (on-chain):

1. Order matching
    (a) The client puts an order called "bid" in the orderbook of the current block
    (b) In order to be able to use the service he pays a fee to the network and the bidding money in escrow (encrypted) on blockchain
    (c) The storer proposes storage space with an order called "ask" and reserves the space
    (d) In the next block the storer accepts a "bid" and signs a "deal" with his private key
    (e) The storer sends the deal directly to the client

(f) The storer writes the deal in the order book of the current block

(g) The client also signs the deal and writes it on the allocation table of the current block

(h) client sends escrow key to storer

2. Settlement

   (a) The storer solves the challenge and retrieves the escrowed money

The work-flow for retrieval data is the following, in two phases (off-chain):

1. Order matching

   (a) The client sends an order called "bid" directly to the storer ("gossip")

   (b) Storer sends an order "ask" directly to the client

   (c) Storer signs the deal and sends it to client

   (d) Client signs the deal and sends it to storer

2. Settlement (either *(a)* or *(b)+(c)*)

   (a) micropayments from the client to the storer as data comes

   (b) client escrows money on blockchain

   (c) Storer claims the payment after having sent data

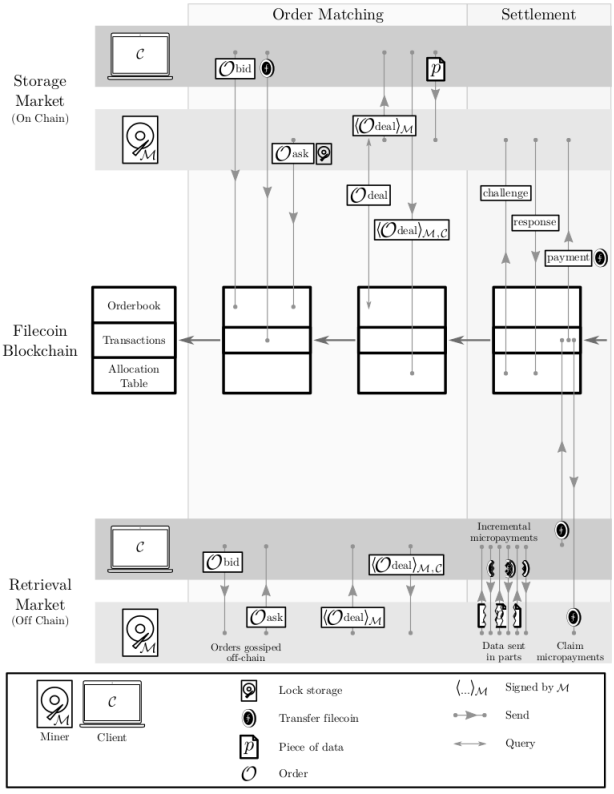


Figure 7: Filecoin[Lab17]

3.4.1 Proofs in Filecoin

The proofs are mechanisms which are used in the filecoin blockchain. Each of the following schemes can be executed and thus mined by anyone in the network.

- **Proof of storage / Proof of replication**

Proof of storage is a scheme used by Filecoin, to prove to the client that the data is still stored. This can be done multiple times. In addition to that, Filecoin implements proof of retrievability, which proves to the client that the data can be retrieved. This is slightly different from the proof of storage, as it could have been stored correctly, but retrieving the data is not possible, as some chunks cannot be accessed. The proof of storage mechanism works exactly like the proof of Storj, where the Verifier sends a list of salt used to rehash the objects to be verified. The difference with Storj is that Filecoin uses zk-SNARKs [BCCT12], such that the verifier does not actually send the challenge but the challenge is calculated transparently. Thus the proof is non-interactive, which is much more practical.

- **Proof of space-time**

Proof of space-time is the scheme that combines time and space. It proves to the client that the data is stored at the time of the challenge. As this proof gives the time-lapse of storage it proves to the client that the data has been stored during all this period. The proof of space-time is implemented as a counter, where the next challenge depends on the counter. Thus the response to the challenge is sequential to the previous one and the time-stamp of the previous challenge is known. Thus, it proves that the storer stored the data in this time-lapse. The following figure shows how the challenge is calculated with the help of the counter.

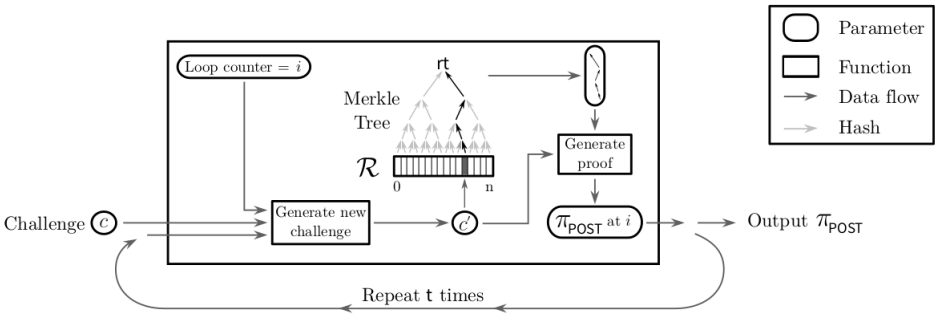


Figure 8: Proof of space-time [Lab17]

4 Conclusion

Generally, the idea of a distributed storage based on blockchain is very promising. The combination of distributed storage and blockchain gives the unique chance to have a verification of the system without any third party. Both systems studied in this report are in operational mode. IPFS seems to be more stable and mature than its competitor Storj. In addition, IPFS seems to be more general than Storj (Storj can be used on top of IPFS). Storj is at its beginning and seems to be a very interesting idea, but the general system seems to rely on some methods which are questionable, such as the naive redundancy specification and the usage of non standard cryptographic procedures.

References

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. pages 326–349, 2012.

[Ben14] Juan Benet. IPFS - content addressed, versioned, P2P file system. *CoRR*, abs/1407.3561, 2014.

[BM07] Ingmar Baumgart and Sebastian Mies. S/kademlia: A practicable approach towards secure key-based routing, 2007.

[Goo18] Google. The go programming language. https://golang.org/, 2018. [online; seen 11 mai 2018].

[Lab17] Protocol Labs. Filecoin: A decentralized storage network, 2017.

[MM02] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric, 2002.

[WBBB14] Shawn Wilkinson, Tome Boshevski, Josh Brandoff, and Vitalik Buterin. Storj a peer-to-peer cloud storage network, 2014.

[Wik18a] Wikipedia, 2018.

[Wik18b] Wikipedia. Kademlia. https://en.wikipedia.org/wiki/Kademlia, 2018. [online; seen 11 mai 2018].