IPXS - IPFS with permissioned files, mutable links, and speed improvements (DRAFT 1)

Archon Technologies, Inc.

October 7, 2019

Abstract

The InterPlanetary eXtended System (IPXS) is a peer-to-peer distributed filesystem that aims to enhance the existing InterPlanetary File System (IPFS) to carry out its mission of connecting all computing devices under the same system of files. IPFS is a widely used peer-to-peer global filesystem. However, it has certain shortcomings that limited its appeal to specific audiences. For example, a lack of built-in permissioning feature for its files limits the ability for any applications with private or semi-private data from using IPFS. Other shortcomings include a lack of mutable links, and performance sluggishness for particular use cases. IPXS aims to address all of these issues by taking advantage of a new architecture redesign as well as new advances in technology, while retaining its backwards compatibility with existing IPFS infrastructure.

1 Introduction

InterPlanetary File System (IPFS) is an attempt to build a global peer-to-peer filesystem, and has succeeded widely within the blockchain community. The typical blockchain storage stack uses IPFS to store data, while having a hash of that data stored on a blockchain smart contract. With this approach, the blockchain verifies the correctness of data in a decentralized manner, and IPFS stores the data in a decentralized manner.

However, despite its popularity, there still remains a lot of room for improvement for IPFS, both to existing users as well as pained potential users. We will illustrate the areas of improvement with the following example uses cases:

1. **Lack of permissioned files:** a blockchain-based social network wishes to create private groups, where group information and messages are only visible to group members. With IPFS, all data are either public to everyone, or encrypted and public to only the key holders. IPFS does not support file permissioning at a group level, so the blockchain-based project has to resort to building its own key-management infrastructure, which not only adds development cost and application complexity, but also introduces a centralized and single point of failure

2. **Lack of mutable links:** a blockchain-based live streaming application wishes to store the live stream decentralized. Live streams require mutable links. In the normal web, live streams are represented by a single URL, where the video data behind the URL constantly changes and updates. However, with IPFS, data is represented by a content-addressed hash. As a result, given a hash, the data behind the hash cannot change. The blockchain-based application cannot store its live stream decentralized using IPFS. They have to resort to using IPNS, which is IPFS's naming system that allows mutable links. However, IPNS is known to be slow, where it is not uncommon for link resolutions to take many seconds. If the blockchain project cannot accept this speed, they have no alternatives for decentralized storage of live streams.

3. **Unscalable uploads:** a blockchain-based archival project wishes to store a large amount of data (1,000,000's of files worth 100's of terabytes of data), but is bottlenecked by the IPFS network. The upload will begin to slow to a crawl beyond 100,000 file uploads at a time for the IPFS public network. The blockchain-based project has to resort to creating a private IPFS node to hold the dataset, because a smaller, pri-

vate network can handle that scale. However, in order to link this private IPFS node to the public network, the project has to set up an IPFS relay node in the public network that relays requests to this private IPFS node. Not only does this increase development cost and application complexity to the application, but it represents a centralized and single point of failure in the form of the relay node.

4. **Slow downloads for long-tail:** a blockchain-based game has a long-tail of decentralized data, and requires the download of the data to be extremely fast. Because their data has a long-tail distribution, there is a large amount of files, and they are accessed very infrequently (perhaps once a week). IPFS download speed is fast if a public gateway is used, such as CloudFlare's IPFS gateway. However, the download speed is only fast for popular files that are already retrieved and cached. For files that are infrequently accessed, the public gateway must use the IPFS protocol to find where the file is being stored, and this peer discovery process is often slow. As a result, accessing a file for the first time or accessing rare files have slow performance. If the blockchain-based game is played in real-time, then this speed will not be an acceptable gaming experience.

5. **Lack of cryptographic proofs and payment:** a blockchain-based application wishes to pay a peer to hold its file with cryptocurrency. In order to have the confidence that the peer is holding the file, the blockchain-based project also requires the protocol to periodically test the peer with cryptographically verifiable proofs. The project cannot use IPFS, since it neither accepts cryptocurrency as payment nor has any cryptographically verifiable proof for any action by the storage peer. Therefore, the project resorts to using FileCoin, which is an incentive layer behind IPFS. However, FileCoin is not built into existing public blockchains such as Ethereum or Neo, but rather built on its own blockchain. The project must integrate an additional blockchain into the application, which increases its development cost and application complexity. Furthermore, FileCoin uses IPFS as its storage layer, not IPXS, so it will have the same limitations as IPFS (lack of permissioned files, lack of mutable links, etc).

InterPlanetary eXtended System (IPXS) was created to extend IPFS with additional features in order to handle these types of use cases. With IPXS, users will be able to store files in a peer-to-peer global filesystem while specifying its permission lists, create mutable links that handles frequently-updated files, and have signficant performance improvements in both upload and download speed.

The goal of this paper is to describe IPXS in detail. Section 2 will give a background by describing IPFS design. Section 3 will describe the IPXS architecture, which will be built on top of IPFS. Section 4 will give an analysis on IPXS and its new feature set. Section 5 will describe our test environments and list benchmarks. Section 6 will conclude this paper.

2 Background

In this section, we will give a very brief overview of the IPFS design.

2.1 Files and Hash

When a file is uploaded to IPFS, it is first divided into multiple blocks before being dispersed into the network. Additionally, the file's multihash is computed by having the blocks all hashed together using a cryptographic hash function.

The file's identifier is its multihash. This detail is important, because the name of the file in the IPFS address space (i.e. the file identifier) is directly tied with the file's content. Therefore, if the file content changes, its identifier must also change. With IPFS, it is impossible to modify the contents of a file while keeping its file identifier the same.

The file identifier is usually text encoded with base58 encoding for things like sharing URLs, but are left as its byte form when stored on smart contracts.

2.2 Routing with DHTs

IPFS uses a Distributed Hash Tables (DHT) to store the list of peers that are holding a particular file. A DHT is a global lookup table that is distributed across a peer-to-peer network. Each peer is only required to hold a small portion of the entire lookup table. If they are queried about a lookup they do not have, they have the ability to forward the query to another peer to carry out the query. This is done in a way that is resilient to known malicious attacks. The average query will contact $\log(n)$ nodes or less, where n is the number of peers in the network.

2.3 Pinning and File Lifetime

In order to continue serving the file, the peer must pin the file to their IPFS client. Otherwise, if the IPFS client is running out of resource, the file will be garbage collected and deleted from the system. Therefore, the file availablity depends on the number of peers that are pinning that particular file.

In practice, IPFS peers tend to altruistically serve files that are popular. Therefore, for popular files, file availability is not a problem, because there will be some peer in the network that would be willing to serve it to the downloader. However, in practice, most files will not be popular enough to attract the attention of these altrustic peers. Typically, these files have only one peer pinning the file, and that would be the uploader themself, or a pinning service that the uploader paid.

2.4 Mutable Links and IPNS

Because IPFS file identifiers are directly tied with the file's content, users cannot update the file content while keeping its file identifier the same. To work around this, the IPFS team created a name-resolution protocol, called InterPlanetary Name System (IPNS), that maps a mutable link with a file stored in IPFS. With IPNS, a file identifier can point to a mutable file, and therefore the file can be updated while having its identifier be kept the same.

In practice however, IPNS name resolution has a slow performance. A latency in the order of seconds is not uncommon. As a result, IPNS user adoption has been hindered, and only a small fraction of IPFS users use IPNS.

2.5 Public Gateways

The IPFS protocol is meant to be used with the IPFS binary, which is compiled from various languages, including Go. In order to get regular web-users and non-programmers to download files from the IPFS network, the IPFS team have implemented public gateways. Public gateways are IPFS peers that are able to serve IPFS data through the HTTP protocol, and they usually have a domain name registered. As a result, a regular web user can access an IPFS file using a normal URL that consists of the public gateway's domain, as well as the IPFS hash of the file. The public gateway then serves this request by using the IPFS binary to download the file in the back end, and then serve the file to the user.

Although public gateways introduces centralization and a single point of failure, its convenience outweighs the negatives. Additionally, users that value decentralization always have the option of using the IPFS binary instead, which is the more decentralized way.

3 IPXS Architecture

In IPXS, there are three main actors:

1. **Nodes:** nodes are entities that store files and participate in one of the DHTs in IPXS. Nodes provide the utility for IPXS, and create the IPXS network. Furthermore, there are three types of IPXS nodes: seeds, peers, and IPFS peers.
2. **Uploaders:** uploaders are entites that upload files into the IPXS network. These entities do not necessary belong in the IPXS network, but instead only uses IPXS. Therefore, these require only an IPXS light-client.
3. **Downloaders:** downloaders are entites that download files from the IPXS network. Like uploaders, these entities do not necessary belong in the IPXS network, but instead only uses IPXS. Therefore, these require only an IPXS light-client.

The main components of the IPXS architecture consist of a **global state database**, a **multi-level DHT**, and an **encoding layer**. We will describe each in detail below:

3.1 Global State Database

3.1.1 Introduction

Every node in IPXS must agree on a global state. IPXS implements this global state database, while keeping the module decentalized, by using a public blockchain. For example, we can store the global state on a smart contract in Ethereum, where every Ethereum node agrees on the smart contract state. Therefore, IPXS nodes only need to run Ethereum nodes or connect to an Ethereum node RPC in order to verify global state data.

The IPXS team plans to implement an IPXS network on Ethereum, and another IPXS network on Neo. The advantage of the Ethereum network is in its level of decentralization, network security, and developer community. Ethereum developer tools will be mature and well-documented, and there are numerous tutorials to learn from. The advantage of Neo is in its faster confirmation times (which allows uploads to be confirmed quicker, an essential user experience to uploaders) and cheaper gas costs. If an uploader wishes to upload many small files, it would be signficantly cheaper

to upload them in the IPXS-Neo network compared to the IPXS-Ethereum network.

3.1.2 Global Seed Registry

IPXS nodes that wish to become seeds must register themselves as seeds in the global state. Seeds will be described in more detail in the Multi-Level DHT section. This section will only focus on seed as it relates to the global state.

IPXS nodes send a RegisterSeed transaction to the blockchain, which requires them to fill seed properties such as their download URL link or IP address. When the IPXS node no longer wishes to become a seed, they send an UnregisterSeed transaction, and their entry is removed from the global state.

The global state will keep track of an ordered list of IPXS seeds and their properties. Therefore, it is guaranteed that all IPXS nodes agree on the order of the IPXS seed as well as their properties.

3.1.3 Global Username Registry

Uploaders in IPXS that wish to upload non-public files, use mutable links, or enable file verification, and downloaders that wish to download non-public files, must both be registered in the global state. A registration consists of sending a RegisterUsername transaction, where they specify user properties such as their public key and a username. Usernames must be globally unique, and is distributed on a first come first serve basis. However, usernames may expire and be transferred to another user.

The global state will keep track of a mapping between a user ID (such as an ethereum address) and its username and public key.

3.1.4 Global Usergroup Registry

Registered users can create groups in the global state by sending a CreateGroup transaction. Groups are a collection of users, moderators, and an admin, with the group creator being the admin by default. The admin has the power to delete the group, add/remove moderators, add/remove users, or transfer the admin role to an existing moderator.

Moderators have to ability to add/remove users to the group. User addition must be approved by both sides (by an admin or moderator of the group, and by the user themselves), resulting in 2 transactions per user addition.

The global state will keep tack of all groups, as well as a mapping between a user ID to its group memberships.

3.2 IPXS Nodes and Multi-Level DHT

3.2.1 Nodes in IPXS

IPXS has three types of nodes:

1. **Seeds:** seeds are IPXS nodes that have greater accountability, and therefore are allowed greater trust. They require a deposit, but can earn crytocurrency by sharing their storage resources. IPXS nodes must be registered in the global state with the following properties:

   (a) **Challenge Frequency:** the frequency that third party authenticators will challenge this peer with cryptographically verifiable tests. A higher frequency is correlated with a higher guarantee of file availability, but a lower frequency is correlated with cheaper costs. Third party authenticators are other IPXS seeds and IPXS peers, randomly selected.

   (b) **Challenge Latency:** the latency the seed promises when completing a challenge or serving a download request. Some seeds may differentiate themselves with fast download speeds (in which case the latency would be small), while some seeds may differentiate themselves with cheap cost. For the latter, they would store the files in an archive-optimized storage unit. This means that downloading from this storage unit would have a significant larger latency.

   (c) **Regions:** a list of one or more geographic regions the servers are located at. This affects the download speed of the file the seed serves, depending on how far away the downloader is from the seed.

   (d) **Minimum Ask Price:** the minimum price the seed is willing to accept to hold and serve the files in the IPXS network. This is the reward the seeds receive for providing utility in the network.

   (e) **BaseUploadUrl:** the base URL uploaders should upload to in order to send the file to this seed

   (f) **BaseDownloadUrl:** the base URL downloaders should download from in order to retrieve a file from this seed. In most cases, the base download URL will be the same as the base upload URL. However, there may be situations in which the seed prefers to set a different base upload URL from a base download URL. For example, once a seed

receives a file from an uploader (who used the base upload URL), the seed may move the file to a separate device that is optimized for file archival, or optimized for the speed of content distribution.

(g) **Deposit:** seeds must lock in cryptocurrency in the form of a deposit. If they are proven to act malicious, or fail too many cryptographically verificable challenges, their deposit will be slashed. If they unregister as seeds, they will get their deposit refunded.

2. **Peers:** IPXS peers have less accountability in the system, but at the same time, it is much easier to join the IPXS network as a peer, as it does not require any deposit or transactions. An IPXS peer simply runs the IPXS node software and connects to the peer DHT in order to be a part of the network. However, note that they cannot earn any cryptocurrency by sharing their storage resources.

3. **IPFS Peers:** IPFS peers are nodes that run the IPFS software, and participate in the IPFS DHT. They already exist, and are currently running the IPFS network.

3.2.2 IPXS networks and Multi-Level DHTs

IPXS has three types of networks:

1. **Seed network:** consists of the group of seeds, coordinated by the Seed DHT. The seed network is designed to be the smallest of the three networks, due to the requirements and accountability associated with being a seed.

2. **Peer network:** consists of the group of peers, coordinated by the Peer DHT. Every node in the seed network is also a part of the peer network.

3. **IPFS network:** consists of the existing IPFS peers, discovered by the IPFS DHT. Every node in the peer network is also a part of the IPFS network.

Note that the three networks are not disjoint. The three different DHTs run in parallel in IPXS. A IPXS seed runs all three DHTs at the same time; an IPXS peer runs only the peer DHT and the IPFS DHT at the same time, and IPFS peers only run the IPFS DHT. As a result, an IPXS seed can serve queries on any file uploaded into the IPXS seed, IPXS peer, or IPFS networks; an IPXS peer can serve queries on any file uploaded in the IPXS peer or IPFS networks; and an IPFS peer can serve queries on any file uploaded in the IPFS network. This allows the IPXS network to be backwards compatible with the IPFS network, while also creating a level of heirarchy between the three groups. The seed nodes have the most preference when it comes to both uploads and downloads. Because the seed network is also the smallest, their file routing resolution will also be the fastest.

3.3 Encoding Layer

3.3.1 Introduction

There are various encoding options for files in IPXS. Some encoding improve file integrity, some encoding improve privacy, and some encoding improve file availability. The different encoding options are specified below:

1. **No encoder:** no encoding is present. The files are uploaded and stored in its raw bytes, unmodified. As a result, no decoding needs to be done after downloading the file.

2. **IPXS container:** the file is stored in an IPXS container, which contains various useful metadata such as checksums, permissioned lists, recovery bits, and signatures for file verification (if the uploader is registered in the global state). The IPXS container must be used if the uploader wishes to upload a permissioned file or a file with mutable links.

3. **Encrypted:** the file is encrypted. The IPXS client will list out possible encryption algorithms (both symmetric and asymmetric), with AES encryption being the default.

4. **Compression:** the file is compressed. The IPXS client will list out possible compression algorithms, with lz4 compression being the default.

5. **Erasure sharding:** the file is encoded into erasure codes before being split into multiple shards. Erasure codes increase the file by a factor of n; however, you only need $\frac{1}{n}$ of the shards (emphasis: *any* $\frac{1}{n}$ of the shards) to recreate the file. This is advantageous for archival use cases, or when the uploader views the peers and seeds as unreliable. Encoding a file with erasure shards allows a certain number of peer to be unavailable while still having the file be available.

Note that these encoding options are not necessary exclusive. An uploader can choose to encode their file with an IPXS container, encrypted, and erasure sharded simultaneously.

3.3.2 Upload Types

There are two types of uploads in IPXS:

1. **Anonymous Uploads:** these uploads do not require the uploader to register, and do not require an upload transaction to be broadcasted. An uploader sends the file to an IPXS seed or peer, or an IPXS node simply broadcasts that they are pinning a file. This is similar to the way IPFS uploads is done right now. Anonymous uploads will only propogate in the peer network and IPFS network, but will not be accepted by the seed network. Furthermore, no encoding is allowed for anonymous uploads.

2. **Registered Uploads:** these uploads require an upload transaction to be validate by the public blockchain. Furthermore, the uploader must be registered in the global state. The advantage of having a registered upload is that the uploader can set the file to be permissioned, use mutable links, or have the ability to upload to the seed network as well as the peer network or IPFS network. Registered uploads have the following fields:

   (a) **File Hash(es):** to verify that the correct file is being uploaded. If the file is erasure sharded, then each shard hash is also listed.

   (b) **Filepath:** if the uploader wishes to create a mutable link, they may specify a filepath to represent the upload, which includes the filename, as well as a directory structure.

   (c) **Encoding Options:** the encoding options the uploader used on this upload.

   (d) **Node Options:** the uploader can specify which nodes should be able to store the particular file, including a random selection strategy, filtering seeds by their properties, or manually selecting individual nodes. If the uploader wishes to keep the manual node selection a secret, then instead of storing the node ID publicly, this field will instead store $hash(nodeID||filehash)$, where $||$ is the concatenation operator.

3.3.3 IPXS Links

IPXS has one immutable link for each anonymous upload, and two links (one immutable and one mutable) for each registered upload. IPXS links generally have the following structure:

username.encodingOptions/fileIdentifier

In this structure, the fields abide to the following descriptions:

1. **username:** the username of the uploader, only available for registered uploads. For anonymous uploads, the username field and the proceeding period are both omitted. Additionally, for registered uploads where the IPXS container encoding is not used and the link is immutable, the username and proceeding period may be omitted.

2. **encodingOptions:** the encoding options of the upload, represented as a text that will look like random alphanumeric characters. This will also specify if the link is an immutable link or a mutable link. For anonymous uploads, this field and the proceeding slash are both omitted.

3. **fileIdentifier:** the identifier of the file. If the link is an immutable link, the file identifier is the content-address hash of the file. If the link is a mutable link, the file identifier is the filepath of the file that the uploader specifies.

As a result, here are some examples of valid IPXS links:

QmT9qk3CRYbFDWpDFYeAv8T8H1gnongwKhh5J68NLkLir6 - hash of the file. Immutable link. Can be generated by anonymous and registered uploads.

h0/QmT9qk3CRYbFDWpDFYeAv8T8H1gnongwKhh5J68NLkLi - encoding and hash of the file. Immutable link. Can only be generated registered uploads.

bob.h2/QmT9qk3CRYbFDWpDFYeAv8T8H1gnongwKhh5J68N - username, encoding, and hash of the file. Immutable link. Can only be generated by registered uploads.

bob.j2/index.html - username, encoding, and filepath of the file. Mutable link. Can only be generated by registered uploads.

bob.n2/temp/images/ipxs.jpeg - username, encoding, and filepath of the file. Mutable link. Can only be generated by registered uploads.

4 Analysis

In this section, we describe new features that IPXS enables, and how IPXS implements them.

4.1 Permissioned Files

4.1.1 Permissioning in an Honest Network

Files in IPXS can be permissioned at the individual level and at the group level. If a file is completely

public, then the downloader does not need to be a registered user. However, if the file is permissioned or private, then the downloader must be registered.

For permissioned files, the IPXS node client simply respects the permissioned list by checking the downloader ID and the global state. Permissioning information is encoded directly in the file. The client reads the permissioning information, and requests the ID of the downloader, then checks the global state for the downloader's group memberships. If the downloader or any of their group memberships belong in the permissioned list, then the download proceeds as normal. Otherwise, the client refuses to serve the download request.

If all nodes are honest in a peer to peer network, then this would be sufficient. However, in practice this is not the case. To make the permissioned files more resilient in the IPXS network, we take advantage of erasure shards. Erasure shards are files encoded into erasure codes, split up into shards.

4.1.2 Permissioning in a Resilient Network

IPXS uses Reed Solomon codes for the erasure code, which means the extra parity bits generated by Reed Solomon Codes by themselves contain almost no information about the original data pieces. Therefore, if the uploader saves only the parity bits as the erasure shards, they have a layer of protection where the downloader has almost no information on the file if they cannot download a sufficient number of shards. This is a similar algorithm to Shamir Secret Sharing, and in fact the math is almost identical.

Let us illustrate this more concretely. An Uploader $\mathcal{U}$ wants to upload a file $\mathcal{F}$ erasure sharded. They encode the file into shards $s_0, s_1, s_2, \ldots, s_n$, and uploads each of them into a different nodes $\mathcal{N}_0, \mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_n$. The erasure shard was parameterized in a way such that $m \leq n$ of them must be retrieved to decode the shards into the original file.

A malicious downloader $\mathcal{D}$ wants to view $\mathcal{F}$, but does not have the permission to do so. Hypothetically, let's say that $\mathcal{D}$ may be able to corrupt one node to act maliciously and serve them the file anyway. Perhaps the downloader is a node themself. In this case, the downloader was able to retrieve one erasure shard. However, unless the downloader is able to retrieve $n - m$ shards, they cannot decode the shards into the original file. In particular, because of the math behind Reed Solomon codes, the downloader has almost no information in what the original file is. The downloader may be able to corrupt more nodes (perhaps bribing with money), but unless they are able to corrupt m or more nodes, they will not gain any information about the original file contents.

IPXS takes advantage of this security to do file permissioning more securely. More precisely, permissioned files are done in the following manner:

1. Uploader must choose a Reed-Solomon erasure shard encoding for the file, along with parameters n and m.

2. Uploader must choose the individual IPXS peers that may hold shards of the file. Every other shard must be held by the seeds. Because seeds have greater accountability, greater trust is given to the seeds to act honestly.

3. For seeds, random download queries will be performed by third party authenticator nodes (seeds or peers randomly selected). If it is proven that a seed has tampered with the code to violate the permissioned list of the file, the third party authenticator will broadcast the proof, and the seed will be punished by having its deposit slashed.

4. The Uploader is ensured that m or more nodes must be malicious in order for the uploader's file to have its permissioned list violated. If the file privacy is important to the uploader, the uploader should choose a number m close to the number n. However, a higher m would make the file availability lower, because only $n - m$ nodes may go down before a file becomes unavailable. Therefore, there is a trade off between file privacy and file availability.

4.2 Mutable Links

Because IPXS is built directly into a public blockchain, IPXS can take advantage of a synchronized global state whereas IPFS could not. This allows IPXS to design a mutable link in a much more straightforward manner.

Only registered uploads can have mutable links. The upload transaction links the mutable link (the filepath) with the immutable link (file hash). Furthermore, the public blockchain maps the uploader's username to their public key, which allows file verification to be done by the downloader. With the combination of these two features, and the fact that IPXS mutable links must contain the uploader's username, designing a mutable link should be fairly straightforward. One thing to note is that the name resolution should be extremely quick, as it is only bottle necked by the node's public blockchain node or public blockchain RPC service, which is a sharp contrast to the consensus mechanism behind IPNS.

4.3 Scalable Uploads and Fast Downloads for Long-Tail

Both of these performance improvements are due to the seed network, which are designed to be small. IPFS has a large network of peers, and although theoretically DHTs scale beautifully, in practice the IPFS network introduces a large amount of delay due to its sheer size. IPXS designed the seed network to be small by introducing a large amount of barrier, which increases the accountability of the seeds. As a result, if a user wishes to upload a large amount data to the network, or download long-tail data with minimal latency, they can choose to upload the files to the seed network to capture the performance gains.

In the benchmarks section, we will demonstrate with experiments the affect the size of the network has on these properties.

4.4 Cryptographic Proofs and Payments

When a registered uploader makes a registered upload, they are allowed to pay seeds to pin the files (or shards) for them. The IPXS protocol creates an incentive mechanism for the seeds to share their resources. Payment is held in escrow, and slowly given to the seeds as they prove they have performed the storage work. Furthermore, if they are proven to be malicious, they will have their deposit slashed by an appropriate amount. Seeds are probabilistic challenged in the following way:

1. The uploader initiates an upload u_i to a specific seed. They separate the upload payload (either the file or a shard) into 1024-byte blocks $\mathcal{B}_0^{u_i}, \mathcal{B}_1^{u_i}, \ldots, \mathcal{B}_{n_i}^{u_i}$, then signs each block. The signatures $\sigma_0^{u_i}, \sigma_1^{u_i}, \ldots, \sigma_{n_i}^{u_i}$ are aggregated into a list.

2. The uploader sends the payload $\mathcal{B}_0^{u_i}, \mathcal{B}_1^{u_i}, \ldots, \mathcal{B}_{n_i}^{u_i}$ as well as signatures $\sigma_0^{u_i}, \sigma_1^{u_i}, \ldots, \sigma_{n_i}^{u_i}$ to the seed.

3. At a random point in time, based on the seed's challenge frequency parameter, the protocol randomly selects certain seeds and peers to issue a challenge to the chosen seed. The seed has time specified by the challenge latency parameter to complete the challenge. The challenge is described in the following steps.

4. The challenge sent to the seed contains indices of 300 random upload index-block index pairs (i, j).

5. The seed must send the 300 blocks corresponding to those upload and block indices, as well as the signatures of those blocks.

6. The challenger verifies that, given the uploader public key as specified by the upload transaction and the file block $\mathcal{B}_j^{u_i}$, the signature $\sigma_j^{u_i}$ matches.

7. A group of challenger undergoes this process at the same time. The seed fails the test if the number of third party authenticators that indicate failure passes a threshold.

Randomly selecting 300 blocks to verify is an efficient way to test the entire inventory of the seed in one batch across files and across uploaders. 300 blocks and signatures would equate to roughly 300 kb of bandwidth, which is fairly small given the potential size of the inventory.

Due to the birthday paradox, regardless of how large an inventory the seed contains (whether it is 1 terabyte or 1 petabyte), if the seeds is malicious and deletes 1% of its inventory, then a third party authenticator only needs to randomly select 300 samples in order to have a 95% probability of catching the deletion. Having this test repeat multiple times by an independently selected group of challengers only strengthens the guarantees.

For readers that wish to see the derivation of these numbers, they can read a mathematical explanation under section 5.1 of the paper Provable Data Possession at Untrusted Stores, by Giuseppe Ateniese, Randal Burns, Reza Curtmola, et al.

5 Benchmarks

TDB

6 Conclusion

Taking advantage of new technologies and novel architectures, such as blockchain and multi-level DHT, we are able to extend IPFS to overcome some of its shortcomings. IPFS already made crucial advances in peer-to-peer storage systems. We hope that IPXS is able to take IPFS and push further its journey to carry out the original mission: to connect all computing devices with the same system of files.