Troposphere: A Decentralized Cloud Storage Network

Matt Gleason

Abstract. Decentralized file storage and sharing is the next step forward in decentralizing the internet, but previous attempts have come with various legal risks or excessive use of blockchains and cryptocurrencies to do a fileserver's job. I propose a solution built on the sound money of Bitcoin, with privacy by default, adequate plausible deniability for node operators, and an efficient marketplace for matching uploaders with storage providers.

1 Introduction

The creation of Bitcoin gave the internet a base layer of decentralized money. The Lightning Network added a layer for fast and cheap payments over payment channels. Nostr added a social layer, where people can control their online identity, communication, and data. Troposphere's goal is to create a layer for decentralized data storage, sharing, and editing. By combining cryptography used in Bitcoin, and the Lightning Network for micropayments, Troposphere will attempt to make it possible for anyone to upload files privately, and anyone to be paid to store those files.

2 System Overview

2.1 Files

Files are padded, encrypted using symmetric encryption, and split into at least 2 fragments of 1024^n bytes. Padding increases the anonymity of Uploaders, by preventing files from being identified by their size. Fragmenting gives Storage Providers a level of plausible deniability when operating under an oppressive regime. Each file is associated with a Bitcoin HD Wallet account, and fragments are stored at addresses derived from the file's account. This allows only those who hold the file's xpub to retrieve the fragments. Only those who hold the file's xpub and encryption key can view the file. Furthermore, those who hold the xpub, encryption key, and xpriv can modify the file.

2.2 Node

Both Uploaders and Storage Providers use the same node software. The node consists of a Bitcoin wallet, a Lightning Node, and a decentralized exchange for file storage. Any node can respond to a query for a fragment, or a request to modify a fragment, and gossip the query to the whole network. The node software could be modified to allow custodial file uploading, by sending a remote node a Bitcoin payment, and a fragment.

2.3 Uploaders

Uploaders convert files into fragments, add give them addresses. Uploaders act as buyers in the exchange. They are encouraged to create 2 or more orders for storage per fragment, so that their file is stored redundantly. After each order is filled, the Uploader can send the fragment to the Storage Provider, and start requesting proofs of storage from the Storage Provider that filled the order, using sha256 hashes of the fragment, with random salts for each hour. After every successful proof of storage, the Uploader pays the Storage Provider the hourly rate for storage over a shared Lightning Network channel.

$$proof = \text{SHA256}(fragment||salt)$$

The Uploader would keep a rolling table of salts and correct proof values for each Storage Provider it is cooperating with, in order to continuously verify data availability, while storing minimal data locally.

2.4 Storage Providers

Storage Providers act as sellers in the exchange. They enforce access control to files by verifying fragment signatures against their addresses. Storage providers can have a range of hardware, and are encouraged to modify this software to best suit their hardware.

2.5 Exchange

The exchange will allow Uploaders and Storage Providers to exchange storage capacity. Each order consists of a price in msats per Kilobyte per hour, a fragment size, and a fault tolerance period in hours. This fault tolerance period is the time used to determine if an Uploader or Storage Provider has dropped their end of the deal.

2.6 Fault Tolerance

If a Storage Provider fails proof of storage challenges for longer than the fault tolerance period, the Uploader can stop paying for storage, and close their Lightning channel. Similarly, if an Uploader stops paying for successful proof of storage challenges, or stops conducting proof of storage challenges altogether, for longer than the fault tolerance period, the Storage Provider can drop the fragment, and close their Lightning channel. If the Uploader fails to uphold their end of the deal for a time shorter than the fault tolerance period, then resumes, they would owe the Storage Provider backpayment for the downtime, before the end of another fault tolerance period starting when the Uploader resumed communications.

3 Conclusion

I have proposed a system for decentralized cloud storage, where indivduals can both host and store data privately and securely. The type of encryption used for files, and the storage mechanism of fragments so long as they can be queried by address, are outside the scope of this specification. I

will make these decisions in my initial implementation, but I would like to allow this project to adopt newer and better symmetric encryption schemes and file storage mechanisms. File synchronization and version control are also outside the scope of this protocol, as they are the job of applications built on top of this layer.

4 References

1. Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf

2. Poon, J., & Dryja, T. (2016). The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. https://lightning.network/lightning-network-paper.pdf

3. Wuille, P. (2012). BIP 32: Hierarchical Deterministic Wallets. https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

4. Palatinus, M., Rusnak, P. (2014). BIP 44: Multi-Account Hierarchy for Deterministic Wallets. https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki