Aether: A New System for Distributed Data Storage

Abstract: We introduce a new distributed datastore Aether. Aether is a cryptographically secured asynchronous pub-sub system with peer-to-peer replication. Aether allows a user to create a distributed data structure only they can modify, but which is publicly accessible and fully replicated peer-to-peer between subscribers. Aether currently supports key-value stores and personal blockchains.

Aether is implemented in Skywire, the Skycoin wire protocol. https://github.com/skycoin/skywire

Aether is a core component in Obelisk and has applications for

- distributed file storage systems,
- distributed communications systems,
- distributed DNS systems,
- ledgers for digital contracts
- distributed markets
- updatable torrents
- systems for distributed consensus

We briefly overview some applications of Aether.

Blob Replication: The Gossip Protocol

- A blob is an array of bytes.
- The byte sequence is SHA256 hashed and this is the id of the blob.
- When a peer receives a new blob, it broadcasts the hash of the blob to all peers it is connected to
- If a peer does not have a blob it is notified of, it requests it

- On connect a peer requests a list of list of hashes for all blobs each peer has

This is a simple Gossip style replication system. Under these rules, if any node in the network publishes a blob, all other nodes will eventually receive a copy. Aether is a gossip protocol that requires each Blob has a valid cryptographic signature from a known public key.

Aether Protocol:

To generate an Aether datastore, an ECC secp256k1 private key and public key pair are generated. The public key is hashed to produce an Aether address (a Skycoin address). This is the address of the datastore.

To publish a byte string to all subscribers, a user hashes the blob and then signs the hash with their private key. They then publish the blob and signature. Peers subscribed to the datastore replicate the blob.

To subscribe to an Aether datastore, a peer inputs the Aether Address of the datastore (a hash of the datastores' public key). Initial Peers replicating the store are discovered using DHT (distributed hash table). Once peers have been found, additional peers are found through PEX (peer exchange).

- When a peer receives a new blob, it broadcasts the hash of the blob to all peers it is connected to
- If a peer does not have a blob it is notified of, it requests it
- On connect a peer requests a list of list of hashes for all blobs each peer has
- Upon receiving a blob, the peer validates the hash and the signature. Invalid blobs are discarded

Aether: Key Value Store

An Aether key value store replicates blob data with the following
- Sig - secp256k1 signature of SHA256 hash of everything following the signature
- Seqw - uint64 incremented every time a new blob is published

- Key - array of bytes
- Value - array of bytes

If a user publishes and signs two entries with the same key, the one with the highest Seqw is valid and the other one is discarded. Therefore keys values can be updated.

Aether: Personal Blockchain

A personal blockchain has blob blocks with the following fields
- Sig - seck256k1 signature of SHA256 hash of everything following the signature
- Seq - block sequence number, incremented each block
- Phash - hash of previous block in chain
- Body - array of bytes, body of block

Digital Contracts With Aether:

Aether personal blockchain enable an elegant implementation of Open Transaction type contracts between two nodes.

Each node runs a personal blockchain. Each block embeds a contract scripting language in the body of each block.

Node A and B enter into a contract. Node A publishes the contract in its blockchain. Node B publishes a signature for the contract. Node A publishes a counter signature to the signature B published.

Example Contracts:
- ASIC miners creates a personal blockchain and issues equities on its chain.
- A sells B a one month American Bitcoin call option for 0.2 Bitcoin, with a strike price of $500 USD
- A sells B a bond with a six month maturity and 10% yield.
- A loans B a loan in Bitcoin at 10% APR, compounded daily, with a minimal monthly payment of the greater of 5% of the remaining principal or 0.2 BTC
- A loans B $500 in Bitcon (at the time the contract was entered) at 10% APR, compounded daily, with payments to be settled in USD with Bitcoin at the price at the time of execution
- A sells B an instrument which converts to $500 USD in Bitcoin at the time of execution (usefully for currency pairs trading and hedging currency risk).

A practical digital contract system requires timestamping, price feeds and other infrastructure. The contract inputs must be public data whose source specified in the contract. Third parties should be able to monitor performance of the contract.

The implementation details of a viable distributed system of digital contracts are extensive and will be outlined in a separate white paper.

Distributed Websites With Aether:

A user takes a website with multiple linked static pages. The URL of each page is the key and the contents of the page is the value. The user uploads the static website into an Aether Key-value store. Now anyone with their Aether address can download and view the Website.

The website will be replicated between all subscribers. The website remains online when the publisher goes offline.

The publisher can update the website. No one can update a page without the private key.

The publisher is difficult to identify, because requests for the website data is served between all subscribers. There is no central server to seize or discover.

Distributed Markets With Aether: Darkmarket Over Aether

Each vendor publishes a distributed website with Aether as a key-value store (as above). The website contains a list of products and services for sale. There is no central server publishing the website and the publisher is extremely difficult to identify.

Distributed Markets With Aether: Distributed Merchant Review System

Aether Personal Blockchains are used for publishing reviews.

Upon entering into a purchase agreement with a merchant, the user enters into a payment agreement (pay $0.4 Bitcoin to Address X). The merchant countersigns the payment agreement and the merchant provides a signed hash of the purchase receipt.

Objectives:

- A user who has not made a transaction with the merchant should not be able to leave a review
- Performance under contracts should be auditable using publicly verifiable information to the greatest extent possible
- The contents of contracts and receipts should be private, unless the user or merchant chooses to reveal the contract publicly to support their position in the review dispute.
- A merchant should not be able to flood their store with fake positive reviews.

If a user is happy with their purchase, they publish their review in their personal blockchain, along with the signed receipt hash. This proves the user had an economic transaction with the merchant but does not reveal what the transaction was or the amount.

If the user is unhappy, they have three options
- publish their signed receipt hash with a review (reveals and proves they had a relationship with the merchant)
- publish their signed receipt hash and payment agreement with a review (reveals relationship with merchant, reveals amount of transaction, proves or disprove that they met their payment obligation, but does not say what transaction was for)
- publish their payment agreement and their receipt with a review (reveals relationship with merchant, whether the payment obligation was met, and what the disputed transaction was for)

To review a merchant negatively, the user must have entered into an agreement with them, which is proved by the signed receipt hash, from which the merchant's public key can be derived. This imposes a cost on a review (a user must have entered into a transaction to publish a valid review).

If a merchant feels the review is invalid because the user did not meet their payment obligation, the merchant publishes the user signed payment agreement, which shows the user did not meet the payment obligation, but does not reveal what the payment was for.

If the merchant feels the review is invalid, because the user gave a negative review for an item they did not purchase, the merchant publishes the receipt and countersigned receipt hash of the transaction, demonstrating what the user bought. (to prevent fake users from buying cheap items to spam negative reviews for other items).

Or a merchant can simply respond to the negative review directly.

Another concern is preventing merchants from creating an army of bots which post fake positive reviews. One method is to validate reviews with a blinded web of trust system.

Distributed Markets With Aether: Blinded Distributed Merchant Review System

A more complex system, blinds the reviews. In the default system, publishing a review reveals the user’s pubkey, proving a user entered into a contract with specific merchant. We want to enforce honest reviews, while leveraging web of trust relationships to validate reviews, while keeping the identity of the reviewer private.

For instance a system might only unblind the identity of the reviewer if the reviewer is in the person’s immediate web of trust by using a shared secret protocol. It might be possible to prove through a third party that a review is by a user with a second degree web of trust relationship with the user, without revealing the pubkey of the reviewer.

The objective of the Aether project is not to implement such a system, but to make it extremely easy for developers to build and experiment with such protocols. Skycoin development is

focused on building the core infrastructure necessary to support these protocols, not upon implementation of extremely specific protocols such as this, even though we have some ideas about how it could be done in practice.

Aether makes publishing data easy. The gateway protocol will make receipts, payments and counter receipts easy.

Updatable Torrent Files With Aether:

A scene group wants to update the contents of a Torrent in the future to add new files as they come out or make changes to old files.

This is currently impossible.

A torrent file consists of a file list. For each file there is a chunk size and a list of hashes for each chunk (or the root hash for the merkle tree of hashes).

In Aether, a scene group generates a key-value store. They store the torrent file in a key in the KV store. When they want to update a torrent, the publisher modifies the key's value with the new file list and chunk list.

The magnet address of the torrent is now Addr:Key, where Addr is the hash of the publishers Aether pubkey and Key is the key the torrent is stored under.

Distributed File Systems With Aether: Dropbox Replacement

You have a set of files you want synced between your computers. You heard that Dropbox reads your files and want to host your own files instead of relying on a third party. You have multiple computers with multiple drives.

You create an Aether key-value store. Each entry in the KV store represents a file. The value field gives the location of the file in the virtual file system and a list of chunks in the file. It may contain other information, such as which computers are storing which chunks.

You may set a policy such as

- I want these folders synced on computer A
- I want these folders synced on computer B

Each folder is encrypted with a secret. To give a third party access to the file, you give them the Addr:Secret, where Addr is the public key hash of the Aether store and Secret is the randomly generated secret the shared folder is encrypted with.

Distributed File Systems With Aether: File Backup

You create an Aether key-value store. Each entry in the KV store represents a file. The value field gives the location of the file in the virtual file system and a list of chunks in the file. It may contain other information, such as which computers are storing which chunks.

You may set a policy such as

- I want 250 GB on computer C's drive to be used as backup
- I want 500 GB on computer D's drive to be used for backup (offsite backup)
- I want at-least two redundant copies of files in folder X to be stored in backup
- I want at-least one redundant copy of files in folder Y to be stored in backup

The policy for each folder is stored as JSON in a key-value pair.

Scenarios:
- If your laptop is stolen, you lose files in folder D stored on laptop, but you have the backups on C and D
- If your house burns down, you lost the files on your laptop and the backups on C, but the offsite backups on D survived for folder X
- If a drive fails, it will automatically redistribute the data to maintain the backup policy

Distributed Consensus with Aether: Obelisk

In Obelisk, each node runs a personal blockchain through Aether. Each node is subscribed to the blockchains of multiple other nodes.

When a block is received from another blockchain, the Obelisk nodes publish the hash of the block received in the next block published. This creates a system of interlinked, timestamps proving that particular nodes and subgraphs were on or offline at a given time.

Obelisk nodes negotiate blockchain consensus and publish their local view of the network to their subscribers. Any algorithm on the graph which achieves global consensus from local consensus on the subscription relationship works.