Distributed File System Built Using Mobile Devices

Whitepaper

Version 0.95 - Draft

June 2021

by

Saurabh Singh,
CEO, Unreal AI
saurabh@unrealai.xyz

Table of Contents

1. Abstract

Exa Protocol is peer-to-peer distributed file system that enables humans to store their data on a decentralized storage network created using the free available storage space on mobile devices. Exa protocol is similar to AWS's S3 service where you can put/get/delete files but it uses multiple nodes and peer to peer technology to achieve the same. With Exa Protocol, there is no single point of failure and offers high speed file retrieval and storage.

With the mobile first approach people who do not have the means to harness the power of decentralization and blockchain will be able to do it for the first time. This in true sense will be democratization of blockchain technology.

A truly distributed files system will exist only with a mobile first approach.

2. Introduction

Decentralization has been the holy grail for achieving Cloud 2.0 and attempts have been made with Filecoin, IPFS etc. However, these protocols were designed to run on personal computers and hence they could not stop centralization of storage when data centres flocked to the network because of the reward.

Exa Protocol uses mobile first approach to decentralize data storage which allows mobile devices to act as public data storage devices. Exa Protocol uses blockchain technology to store files in the network and incentivizes those who contribute their free storage on their mobile devices in exchange for Exa Tokens.

This paper is heavily influenced by IPFS Whitepaper and uses similar protocols which are optimized for mobile first approach.

3. Background

This section reviews important properties of successful peer-to-peer systems, which Exa Protocols refers and combines similar to IPFS.

3.1 Distributed Hash Tables

Distributed Hash Tables (DHTs) are widely used to coordinate and maintain metadata about peer-to-peer systems. For example, the BitTorrent MainlineDHT tracks sets of peers part of a torrent swarm.

3.1.a Kademlia DHT

Kademlia is a popular DHT that provides: Efficient lookup through massive networks: queries on average contact ⌈log2(n)⌉ nodes. (e.g. 20 hops for a network of 10, 000, 000 nodes).

- Low coordination overhead: it optimizes the number of control messages it sends to other nodes.
- Resistance to various attacks by preferring long-lived nodes.
- Wide usage in peer-to-peer applications, including Gnutella and BitTorrent, forming networks of over 20 million nodes.

3.1.b Coral DSHT

While some peer-to-peer filesystems store data blocks directly in DHTs, this “wastes storage and bandwidth, as data must be stored at nodes where it is not needed”.

The Coral DSHT extends Kademlia in three particularly important ways:

- Kademlia stores values in nodes whose ids are“nearest” (using XOR-distance) to the key. This does not take into account application data locality, ignores “far” nodes that may already have the data, and forces“nearest” nodes to store it, whether they need it or not. This wastes significant storage and bandwidth. Instead, Coral stores addresses to peers who can provide the data blocks.
- Coral relaxes the DHT API from get_value(key) to get_any_values(key) (the “sloppy” in DSHT). This still works since Coral users only need a single (work- ing) peer, not the complete list. In return, Coral can distribute only subsets of the values to the “nearest” nodes, avoiding hot-spots (overloading all the nearest nodes when a key becomes popular).

- Additionally, Coral organizes a hierarchy of separate DSHTs called clusters depending on region and size. This enables nodes to query peers in their region first, "finding nearby data without querying distant nodes" and greatly reducing the latency of lookups.

3.1.c S/Kademlia DHT

S/Kademlia extends Kademlia to protect against malicious attacks in two particularly important ways:

- S/Kademlia provides schemes to secure NodeId generation, and prevent Sybil attacks. It requires nodes to create a PKI key pair, derive their identity from it, and sign their messages to each other. One scheme includes a proof-of-work crypto puzzle to make generating Sybil expensive.
- S/Kademlia nodes lookup values over disjoint paths, in order to ensure honest nodes can connect to each other in the presence of a large fraction of adversaries in the network. S/Kademlia achieves a success rate of 0.85 even with an adversarial fraction as large as half of the nodes.

3.2 Block Exchanges - BitTorrent

BitTorrent is a widely successful peer-to-peer file sharing system, which succeeds in coordinating networks of un- trusting peers (swarms) to cooperate in distributing pieces of files to each other. Key features from BitTorrent and its ecosystem that inform IPFS design include:

- BitTorrent's data exchange protocol uses a quasi tit- for-tat strategy that rewards nodes who contribute to each other, and punishes nodes who only leech others' resources.
- BitTorrent peers track the availability of file pieces, prioritizing sending rarest pieces first. This takes load off the seeds, making non-seed peers capable of trading with each other.
- BitTorrent's standard tit-for-tat is vulnerable to some exploitative bandwidth sharing strategies. PropShare is a different peer bandwidth allocation strategy that better resists exploitative strategies, and improves the performance of swarms.

3.3 Self-Certified Filesystems - SFS

SFS proposed compelling implementations of both (a) distributed trust chains, and (b) egalitarian shared global namespaces. SFS introduced a technique for building Self- Certified Filesystems: addressing remote filesystems using the following scheme

```
/sfs/<Location>:<HostID>
```

where Location is the server network address, and:

```
HostID = hash(public_key || Location)
```

Thus the name of an SFS file system certifies its server. The user can verify the public key offered by the server, negotiate a shared secret, and secure all traffic. All SFS instances share a global namespace where name allocation is cryptographic, not gated by any centralized body.

4. Protocol Design

Exa Protocol is a distributed file system which synthesizes successful ideas from previous peer-to-peer systems, including DHTs, BitTorrent, and SFS.

Exa Protocol is peer-to-peer; no nodes are privileged. Exa protocol nodes store Exa Protocol objects in local storage. Nodes connect to each other and transfer objects. These objects represent files and other data structures. The protocol is divided into a stack of sub-protocols responsible for different functionality:

- **Identities** - manage node identity generation and verification.
- **Network** - manages connections to other peers, uses various underlying network protocols. Configurable.
- **Routing** - maintains information to locate specific peers and objects. Responds to both local and remote queries. Defaults to a DHT, but is swappable.
- **Exchange** - Block exchange protocol (BitSwap) that governs efficient block distribution. Modelled as a market, weakly incentivizes data replication. Trade Strategies swappable.
- **Shards** - Shard represent the smallest piece of the object stored in the node. Described in 4.1.

- **Objects** - Represents the actual file which is derived by combining multiple shards from different nodes. Described in section 4.1.
- **Naming** - A self-certifying mutable name system.

The above terminologies have been inherited from IPFS apart for "**Shards**" which has been introduced for mobile specific environments to maintain data security and integrity.

4.1 Shards and File Storage

When a file is ready to be stored in the network, it is split into multiple segments, called Shards. The shards are encrypted at the client level, are linked together (like a blockchain) and finally the individual segments are stored in multiple peers. The client has a complete hash table that allows it to get the individual shards from the network back. The peers for storage are selected by a reputation score - refer to section 4.4.

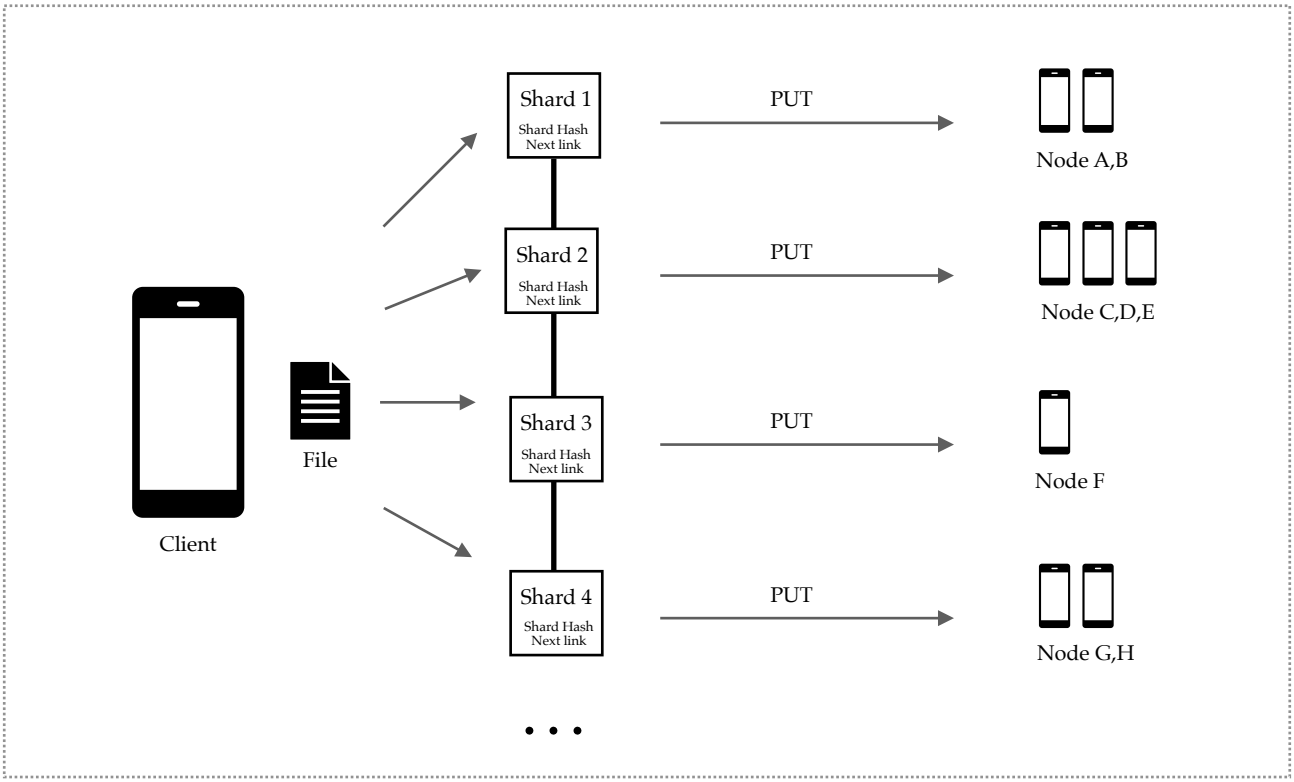

Fig 1: The file is broken down into shards, encrypted and linked by the client and saved on multiple nodes in the network. One shard can be stored on multiple nodes for redundancy in case one node goes offline.

The shards are encrypted using zero-knowledge key encryption so even we will not know the key that is wrapping your data making it secure.

4.1.1 File Storage Algorithm

```
//ready the file to be uploaded
file = getFile()

//encrypts the file using peer specific private key
encrypted_file = file.encrypt(privateKey)

//break the files into segments called shards
shards[] = encrypted_file.createShards()

//scans the network and finds nearest peers with good reputation
peers = getNearestPeers()

//upload the segments and store the peer address
for (shard,peer) in (shards,peers) {

    shard.upload(peer)

}
```

Fig 2: File storage algorithm

4.2 Routing

In a P2P network, with millions of peers, each peer cannot hold a complete reference of all other peers in the network. Such a table would take up a lot of resources on each peer, and would also be almost impossible to keep up to date, as peers join and leave the network. Instead, a routing table is maintained by each peer that contains references to a subset of peers in the network.

Peer referencing is typically achieved by assigning randomly distributed identifiers (GUIDs) to each peer in the network. It is mapped to the node's IP address and port. The GUID is typically designed to work well with the chosen routing table design of the given P2P network routing algorithm. For instance, Chord and Kademlia use a number from 0 to N as GUID, where N is the maximum number of peers the network can contain. Typically a GUID of 64 bits, 128 bits, 160 bits or 256 bits is used.

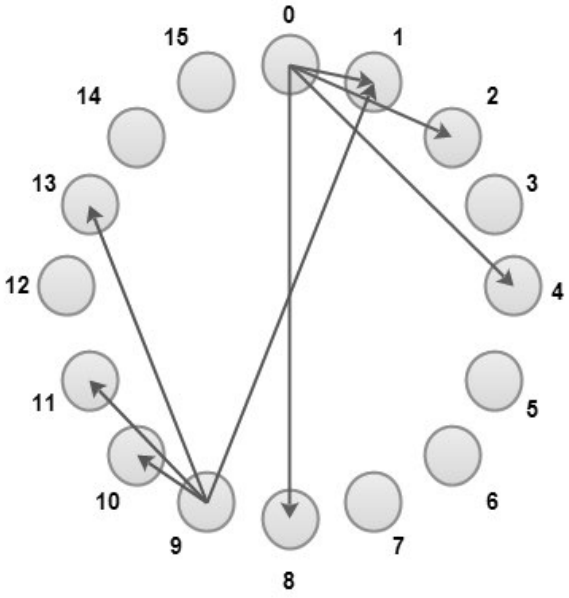

Fig 3: This is peer routing table in a Kademlia P2P network showcasing routing tables for nodes with GUID 0 and 9

4.2.1 Peer Searching

The routing tables will organize the peers it knows according to their peer GUIDs in a way that assigns a virtual location to each GUID and thus its corresponding peer. Peers will then contact peers closer and closer to the peer it is looking for, each time asking for the closest peer to the peer it is looking for.

4.2.2 Joining

The first peer to join the network apart from the boot peer, will connect to the boot peer and send a "join" message. The boot peer responds with a new GUID to the joining peer. After getting a GUID, the joining peer requests to get a copy of the boot peers routing table then use this routing table to find the peers it should have in its routing table.

4.2.3 Management

To keep the routing tables up-to-date despite peer crashes, each peer in the network must periodically search for the correct peers to store in its routing table. If a peer does not respond, that peer should be removed from its routing table.

Similarly, if a closer peer than the peer currently kept in a given cell in the routing table, has somehow joined the network without notifying the updating peer, such a closer peer would be

found during the routing table management process. Such new, closer peers can then be added to the routing table.

4.2.4 Leaving

When a peer no longer wants to be part of a P2P network, it will send a "leave" request to all peers in its routing table. Thus, each of these peers can remove the leaving peer from their routing table. Once a "leave" message has been sent to all peers in the leaving peers routing table, the peer can safely close down all network connections, and shut down.

4.3 File Retrieval

During the file PUT process, the client shards the file and stores the individual shard hash in the hash table. The hash table is a mapping of the shard and on which node/nodes the shard was stored. After that peer to peer protocols, such as BitTorrent, are used to download all the pieces of the file. Once all the pieces of the files are retrieved, they are decrypted and stitched together to produce the original file.

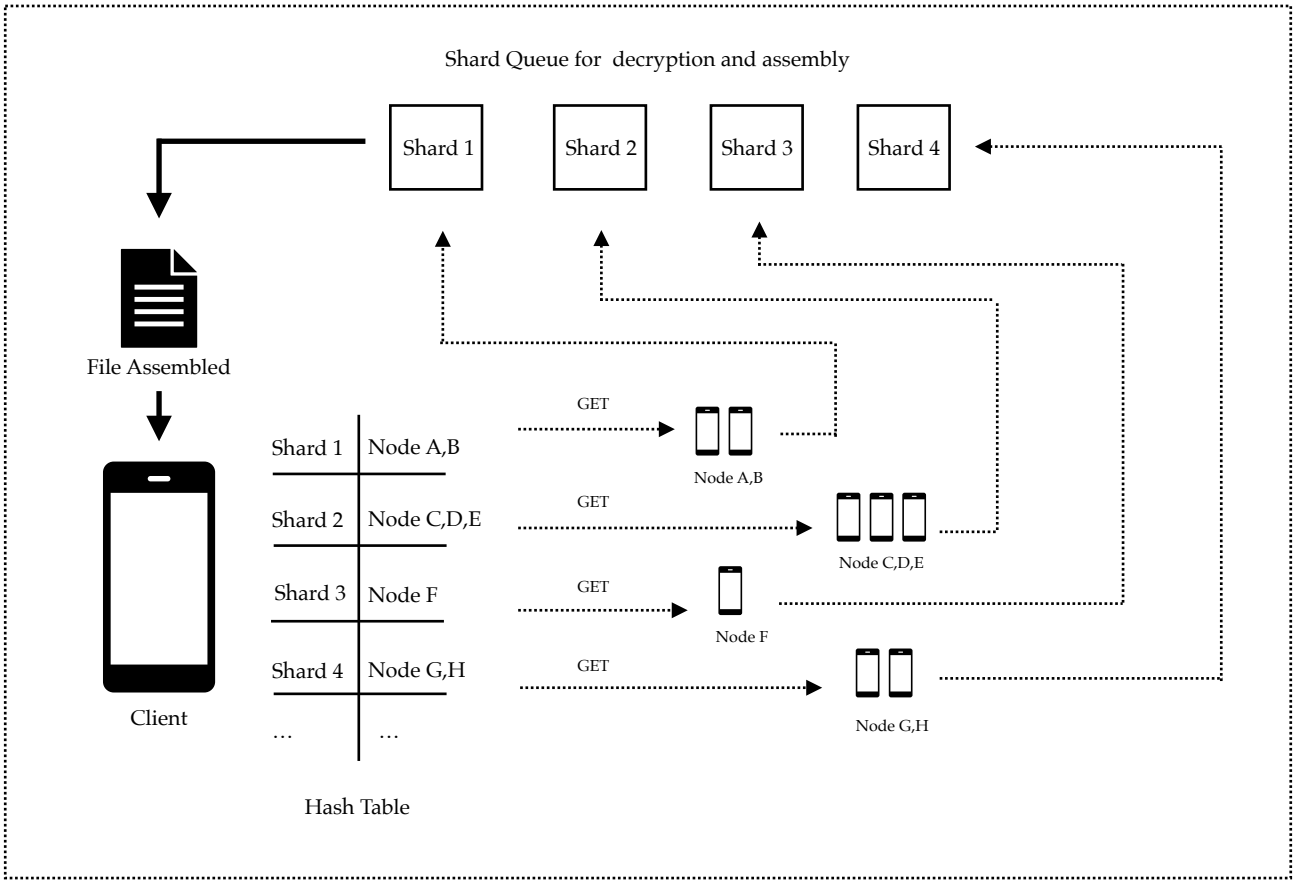

Fig 4: File retrieval process of Exa Protocol.

4.3.1 File Retrieval Algorithm

```
//load hash table
hash_table = getHashTable()

shards[]

//get all the shards and add to shards array
for peer in hash_table.peers {

    shards.append( peer.getShard() )

}

//assemble the shards
encrypted_file = shards.assemble()

//decrypt the file using peer specific private key
file  = encrypted_file.decrypt(privateKey)
```

Fig 2: File storage algorithm

4.4 Node Selection for shard storage

The selection of a node in a decentralized system will be dependent similar to a distributed system where we try to select nodes in different regions to ensure that even if one region suffers a loss of the shard the other region can help it recover. The initial redundancy factor for this one will be 4 - that is 4 replicas of the data will be stored on 4 different peers.

Many factors will be taken into account from the point of a tenant. There will be few factors we'll be keeping track of on the platform to ensure the maximum availability of the file to the user and the most efficient usage of the available storage.

Few of the factors include:

- Uptime
- Network Performance
- GeoLocation

- Type of Device
- Age of Device

Uptime - U

Uptime U is the ratio of time the node was connected to the network to the total time since the node has registered on the network.

Network Performance - Np

Network performance N refers to the average bandwidth of the node while downloading or uploading a shard. The round trip response time, or ping, is also taken into consideration.

GeoLocation - G

A node that is closer to the client is preferred. Nodes that are nearby have a higher probability of low ping. A coarse GeoLocation (~150Km radius, 1ms theoretical ping) is recorded for each peer in the network.

Type of Device - T

A device with a LTE or 5G connection is preferred over a device with just Wifi. A device with WiFi has a higher probability of going offline.

Age of Device - A

Mobile devices typically lasts for 2-3 yrs which means that a newer device will be given a priority.

This results in a reputation score R which is associated with each node during the time of storage of the shard. The Protocol prioritizes devices that have high reputation score to store their shards.

$$R = f(U, Np, G, T, A)$$

Our algorithm won't choose nodes randomly but will be affected by the above key metrics which we will call User/Customer Confidence. Better confidence will ensure these nodes are chosen more frequently in the network

4.4.2 Shard Redundancy Factor

The initial redundancy factor for this one will be 4 - that is 4 replicas of the data will be stored on 4 different peers. As the redundancy factor is 4, 1 will be primary storage and the other 3 will be replicas and the cluster manager can rank them accordingly through an algorithm. We'll put various nodes in different regions to ensure the file is available even during a point of failure. A redundancy factor of 4 ensures a file availability of 99.97% at all times.

When we say initial replication factor of 4 that means there is 1 primary node and 3 replicas. We are keeping them in separate regions as discussed in the previous sections. We are keeping these replicas as this is a decentralised system and we want it to be like a distributed system without the centralisation aspect of it.

With our initial analysis, we realised 4 is an optimal number and with further testing, we can raise it to 5,6 or reduce it down to 3 as well based on the system requirements.

4.5 Node/Peer Types

Exa protocol will introduce two special types of labels for nodes with the possibility of multiple nodes having this position.

4.5.1 Cluster Manager

This will just be a log storage of the whole global distribution pattern of data storage and node status. Along with the deployment of one primary node by exa, multiple replicas of this will be chosen from the available nodes in the network based on confidence scores.

Along with the primary function of data storage this node will keep monitoring the health/ availability of the nodes and raise requests for appropriate action to the concerned nodes. (Local cluster heads)

Overall this node will not be the decision maker but just an observer and fair logger of the data. It may also request a local cluster head to replace a shard storage node if it's been down beyond a certain point, although the request has to be executed by the local cluster head and reported back.

4.5.2 Local cluster heads

To keep the decentralized nature of the network intact, local cluster heads will be selected from every region based on the user confidence scores and with threshold of the same node not being selected as a cluster head beyond a certain number of 'k' rounds with the exception of that node being the only node in the cluster.

These cluster heads will choose the storage pattern of the file and send the logs to the primary cluster manager along with a unique key retrieved after storage of the shards.
These shards can be verified by all the cluster managers across different replicas to ensure a local cluster head is not providing false information about the storage.

4.6 Data Security

For every file stored in the Network, Exa protocol offers 3 levels of security:

- Each file is encrypted with military grade protocols (AES 256 key). The encryption keys are always kept with the peer. This offers Zero Knowledge Encryption - refer to section 4.51.
- It is then split into shards, that are stored on multiple peers with added replication for redundancy and continuous uptime. This means that no peer has the complete file and no knowledge about the other shards.
- All these shards are spread across Exa Protocol's Zero Knowledge Network.

4.5.1 Zero knowledge encryption

Zero Knowledge Encryption means that no one, except you (not even the service provider) can access your secured data. This is achieved by private keys that are unique to the peers. The network has zero knowledge about the keys and hence no once expect the owner can decrypt the file even if they have all the shards assembled.

Also, since the files are encrypted before leaving your device, even someone listening on the network won't be able to decrypt the file. Same is true when downloading the files from peer. They are decrypted on your device and assembled. This ensures end-to-end security for all your data.

5. Network Viability

The current generation Blockchains such as Ethereum offer a low TPS (Transaction Per Second) of around 12 which is ~5000 times lower for a data storage infrastructure. A storage infrastructure with a TPS of over 65000 is required at scale. Thus a blockchain with high TPS (preferably greater than 65000) is required to handle the shard logs.

Therefore, we will introduce our own high TPS blockchain Exa Mainnet which uses PoST (proof of space and time) consensus. PoST is PoS (proof of space) with sequence of checks overtime along with PoRep, PoRepD. PoRepD (proof of replication before dying) is introduced for mobile environments - explained in section 5.4.

5.1 PoS - Proof of Space

Proof-of-Space (PoSpace) schemes allow prover *P* to convince verifier *V* that *P* has spent some storage resources. PoSpace schemes are PoW (proof of work) schemes where the expended resource is not computation (CPU instructions) but rather storage space. In a sense, a PoS scheme is also a PoSpace, since a PoS implies the use of storage resources.

5.2 PoST - Proof of Spacetime

Proof-of-Spacetime (PoSt) schemes allow prover *P* to convince verifier *V* that *P* has spent some "spacetime" (storage space used over time) resources. This is a PoSpace with a sequence of checks over time. A useful version of PoSt would be valuable as it could replace other PoW schemes with a storage service. This work introduces such a scheme, based on sequential PoReps.

5.3 PoRep - Proof of Replication

Proof-of-Replication (PoRep) schemes (this work) are another kind of PoS that additionally ensure that P is dedicating unique physical storage to storing *D*. *P* cannot pretend to store *D* twice and deduplicate the storage. This construction is useful in Cloud Storage and Decentralized Storage Network settings, where ensuring a proper level of replication is important, and where rational

servers may create Sybil identities and sell their service twice to the same user. PoRep schemes ensure each replica is stored independently. Some PoRep schemes may also be PoRet schemes.

5.4 PoRepD Proof of Replication before Dying

Since mobile devices have an average lifespan of 2.5 yrs it is imperative that the data from mobile device is copied to a healthy peer before its life ends. PoRepD is a scheme where prover P convinces verifier V that the data has been replicated to a healthy peer before the device is deactivated.

6. Coin Economy

Exa Protocol Coin - EXAP - are the means to use the network to store or retrieve files. For a client to store a file in the network they will have to send EXAP to our smart contract. Clients who provide the storage space (Nodes) on the network will be rewarded with EXAP which are released upon successful storage of the shard.

Nodes can also use the coins to store their own files in the network. With the release of Exa APIs people with the coins will be able to create DAPPs that run using the storage of millions of mobile devices with no reliance on services like AWS or GCP. Later, with the release of Exa Compute the users will be able to host full fledged Web Apps (WDAPP) using the tokens. The nodes who provide the compute will be rewarded with EXAP. Explained in Section 6.

Here's how the coins will be utilized:

6.1 Coin transaction framework

The coin transaction is based on the amount of data that is transferred to the clients and the own data stored or received from the network. This is called the data debt.

$$dataDebt = \frac{dataExternal}{dataOwn + 1}$$

- ***dataExternal*** is the amount of external data that is sent or received by the node.

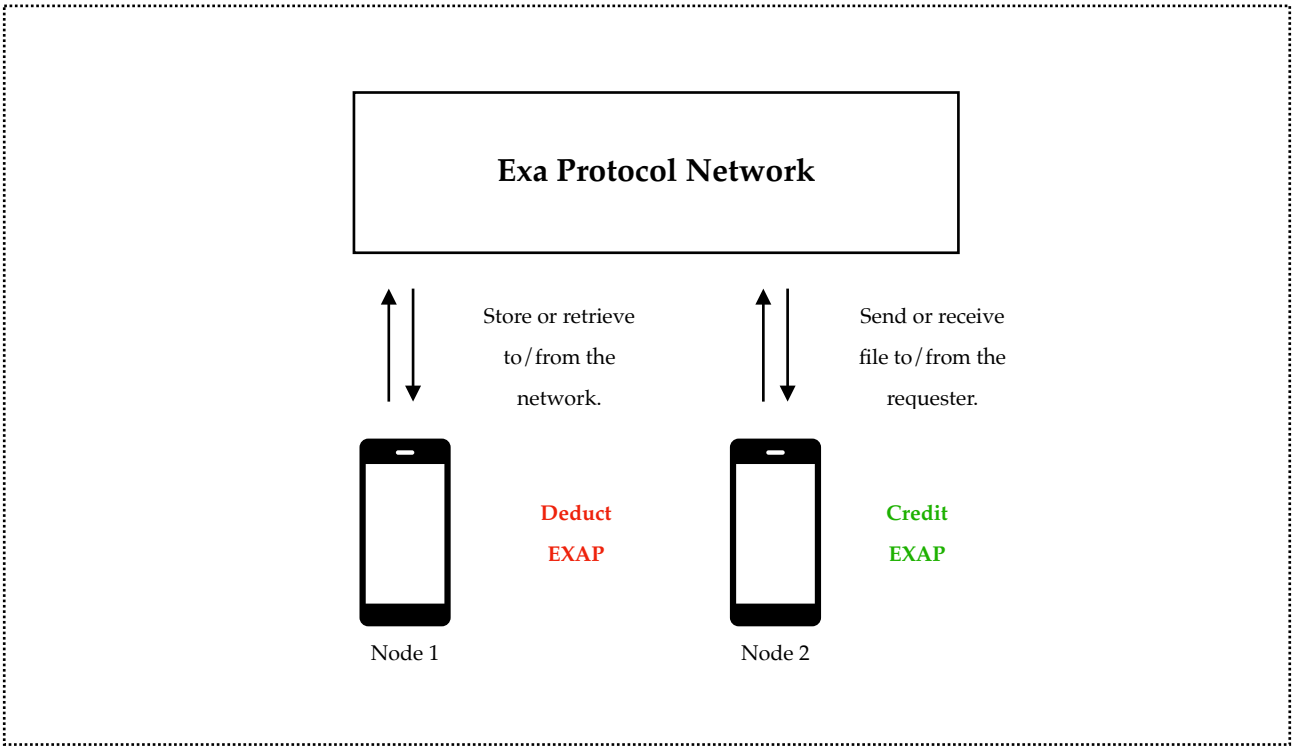

Fig 5: Coin Economy.

- ***dataOwn*** is the amount of own data sent or received from the network.

The ***dataDebt*** is calculated on a daily basis and if,

$$dataDebt > 1$$

Credit EXAP Coins to the node.

$$dataDebt < 1$$

Deduct EXAP Coins from the node. For ***dataDebt =1*** no tokens are credited or debited.

The Network consensus mechanisms such as PoST, PoRepD govern the coin allocation.

7. Future

The idea behind Exa Protocol is the advancement in mobile technologies that make them at par with a personal computer. Majority of the people on this planet own a mobile phone and not a personal computer. They have been kept out from the decentralization revolution that we are witnessing. Exa Protocol will be a stepping stone to democratize technologies such as blockchain.

IPFS has successfully demonstrated how a distributed file system can work but the scale that it needs can only be harnessed with a mobile first approach.

Exa Protocol has future plans to launch Exa Compute to decentralize the computing resources from the corporates.