Moracle: Decentralized data retrieval and storage

Jackson Roberts
jacksonroberts@protonmail.com
moracle.network

Abstract

A decentralized oracle/proof-of-existence system addresses the issues of centralization and reliability that are often faced with centralized oracle services that currently exist. This document proposes a solution that utilizes network consensus via blockchains taking place within segregated data chains. Such isolated data chains allow nodes to participate without subscribing to the entire network. In the event of downtime, nodes within a data chain can easily catch-up by syncing their missing records/blocks with their peers. This allows for a stable and large networking that isn't taxing on individual nodes. Additionally, the ability for anyone to participate in the network creates a perfectly competitive market, resulting in extremely low fees.

1. Introduction

Due to the deterministic nature of blockchain technologies, accessing external information requires the use of a third-party data service, often referred to as an "oracle". These "oracle" services prevent external data from changing or becoming unavailable which ensures the integrity of a blockchain is maintained. However, all current versions of these services are centralized and disallow parties other than themselves from participating in the process of retrieving, storing, and serving information.

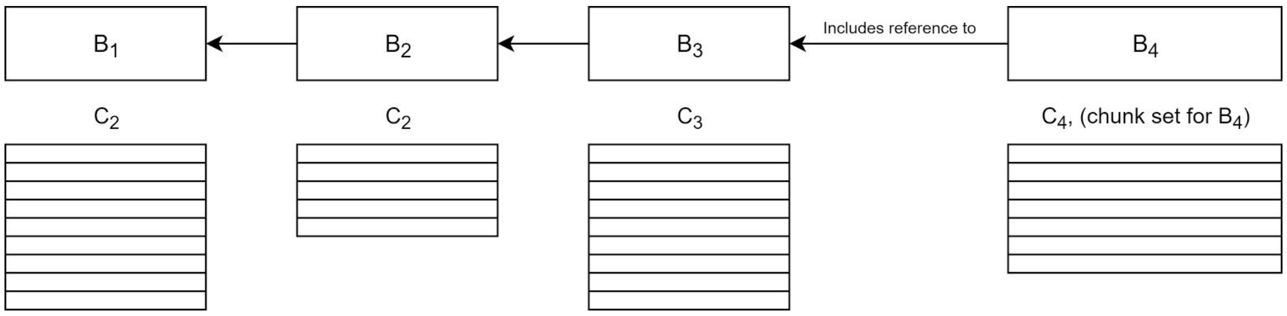
To address this, a information delivery and storage service based on network consensus rather than trust in an authority is needed. Running in a Lisk sidechain, such a network would allow any node to participate, volunteering their drive space and bandwidth in order to provide data retrieval, storage, and delivery services. Such contributions would be incentivized by rewarding side-chain tokens to participating nodes. These tokens would act as currency for the entire decentralized oracle network. Payment would be required for any data storage requests, preventing spam or denial-of-service attacks. However, due to anyone being able to participate in the network and set their own fee thresholds, prices would ideally remain extremely low.

2. Design

The blockchain structure within such a decentralized oracle system is different from the traditional design seen within Bitcoin and other cryptocurrencies. In a traditional blockchain, cryptographic hashes exist to solidify a group of transactions, with each block in the chain adding additional security. In the blockchain of our decentralized oracle, the network is segregated into multiple chains: a transaction chain (of which there is only one), and data chains (of which there are an arbitrary amount). As a single data chain would require immense drive space and memory to participate in, the Moracle network consists of many independent data chains, which can also be referred to as "streams". Blocks of data chunks within each data chain are hashed only to ensure integrity and prevent tampering.

2.1 Data chain

A data chain is defined as a unique blockchain upon which information is stored. Each data chain is structurally identical, all capable of holding any arbitrary data. Data blocks are the primary structure within every data chain. A block is defined as a set of data chunks (as defined in Section 2.1) plus a hash and delegate signature.



Each data chain is identified using a single 64 bit integer, which allows for an extremely large number of unique chains. Most public decentralized applications will be encouraged to make use of a popular data chain.. However, as a data chain is defined simply bty an arbitrary identifier which denotes the data blockchain to search or store within, any application will be able to make use of any chain, provided there are active nodes on it. As a new data chain doesn't cost anything to create, large applications will be capable of easily segregating their data onto its own chain if desired.

By design, participation in a chain requires storing the full data blockchain. This is so all nodes will be able to serve all data, and no data will be lost over time. However, this massively increases the amount of storage space required to participate compared to solutions which nodes can opt to store as much data as they can[1]. By segregating the entire data library of the network into smaller sections via segregates data chains, individual nodes will be able to participate in the network more easily.

2.2 Data chunks

In Moracle, we define each individual piece of data as a “chunk”. These chunks are represented in the form:

$$\text{chunk} = (\mathrm{H}(\text{data}, \theta), \theta, S, T, I, \text{data})$$

where H(data, θ) represents the SHA3-512 (Keccak) hash of the data plus the ahead-of-time timestamp (Section 2.2), θ represents the ahead-of-time timestamp, T represents a real-world timestamp, I represents the transaction ID for the chunk creation (Section 2.4) , and S represents the desired data chain for the request to be processed on. These individual chunks vary in size, and are the data structures contained in data blocks.

2.3 Ahead-of-time Timestamps

When retrieving a stored copy of an internet query result, it’s extremely important to be able to specify which version of that result is required as external resources are extremely subject to change with time. However, it is not important to use real-world time when making that request. In the use case of oracle, the only important thing is to ensure you’re requesting the same result as when it was originally queried. To accomplish this, we make use of an ahead-of-time timestamp which is a unique key that functions as both an identifier, but a proof-of-existence tool.

Take the following example of a possible query to the Moracle API from a theoretical JavaScript decentralized app:

```
var result = MoracleCall({querytype: 'HTTPRequest', endpoint:
'http://exampleapi.com/endpoint?key=foobar'})
```

This example assumes the use of a theoretical API wrapper for Moracle that exists in a Lisk sidechain. An ahead-of-time timestamp must be computable in a way that is fully deterministic, and can thus be recomputed when the result needs to be retrieved again. The primary intended use case of Moracle as a decentralized oracle service for Lisk decentralized apps; thus, an ahead-of-time timestamp could simply be represented in the form:

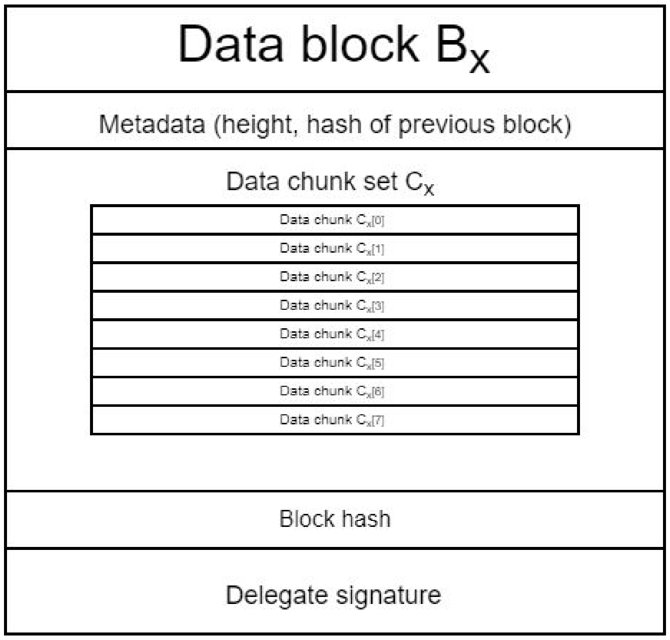
$$\text{stamp} = \mathrm{H}(R, A, I)$$

in which H is the SHA3-512 function, *R* is the JSON representation of the object passed to `MoracleCall`, *A* is the address of the issuer of the transaction, and *I* is the current block height within the Lisk sidechain the decentralized app is running in. This design allows the ahead-of-time timestamp to be computed immediately prior to inclusion in a block within a sidechain.

However, this is only one of many possible ahead-of-time timestamp generation techniques. Decentralized apps which only rely on a few external data points (such as a cryptocurrency exchange rate) and don't allow users to make arbitrary external requests could simply agree to use the hash of the request plus the real world time-stamp rounded to the nearest 1000 seconds. All that matters is that all members of a network are capable of determining the correct ahead-of-time timestamp to use when making requests.

2.4 Data blocks

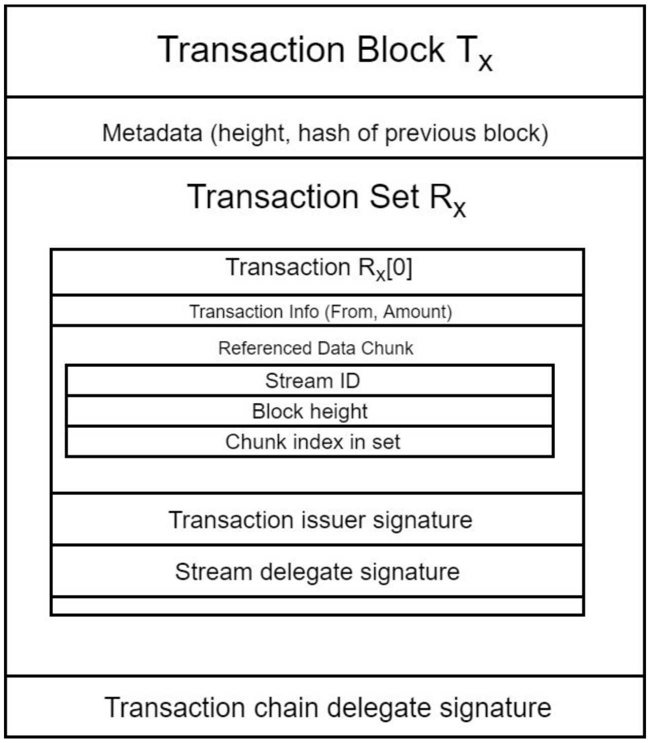
Data blocks are the primary structure within every data blockchain. A block is defined as a set of data chunks (as defined in Section 1.1) plus a hash and delegate signature.



As blocks are not generated using an intrinsically worthless proof-of-work method (Section 2.5), the timing of block generation can be changed on demand according to protocols defined in the network specification. An advantage to this is that an empty block will never be created. In the event that no data chunks were created in a 10 minute block time window, the block will simply be skipped and the delegate will not create a new block.

2.5 Transaction blocks

Transaction blocks are the primary structure within the transaction blockchain. Each block is defined as a set of transactions (as described in Section 2.5), plus a hash and delegate signature.


Transaction Block T_x
Metadata (height, hash of previous block)
Transaction Set R_x
Transaction $R_x[0]$
Transaction Info (From, Amount)
Referenced Data Chunk
Stream ID
Block height
Chunk index in set
Transaction issuer signature
Stream delegate signature
Transaction chain delegate signature

2.6 Other structures

In addition to the already defined data chains (Section 2.1), data chunks (Section 2.2), data blocks (Section 2.4), and transaction blocks (Section 2.5), Moracle also implements the following structures:

1. Pending data storage request: a request sent to nodes of any given data chain. Includes requester signature, data in full, and the amount of fees to be paid.
2. Data storage receipt: a message sent from a data chain delegate to the transaction chain. Includes all data from the pending data storage request (except for the full original data), as well as the signature of the sidechain delegate and the information on where the data is now stored in a data chain.

2.7 Moracle Tokens

Abbreviated as MRCL, Moracle Tokens are used to pay for every data storage request on a data chain. New coins are forged by transaction chain delegates and shared with data chain delegates through the processes described in Section 2.6.

2.8 Transaction lifecycle

A. For a transaction involving a data storage request:

1. User sends request for data storage on a given chain, pending request (Section 2.6.1) is released only to that chain.
2. Data chain delegate processes request, mines data block including the new transaction, releases documentation of this into the transaction chain via a data storage receipt (Section 2.6.2).
3. Transaction chain delegate receives the data storage receipt (which is distributed to all main chain nodes), and creates a matching transaction record (Section 2.5, Transaction $R_x[0]$).
4. When the block containing a reference to the requester's data, the transaction is complete, and the fees are distributed between both the transaction chain and data chain delegates. New MRCL are created as a result (in a process identical to the Lisk forging process), and the sum of the newly forged MRCL and fees is distributed between the data chain and transaction chain delegates that were involved in the transaction.

B. For a simple transaction involving the transfer of MRCL (Moracle tokens) from one address to another the following simplified procedure is performed:

1. User sends signed and completed transaction record with "stream (data chain) delegate" and "referenced data chunk" fields left blank.
2. Transaction chain delegate receives the transaction record and inserts it into a block.
3. The transaction is completed, and fees and newly forged MRCL are claimed by the transaction chain delegate.

2.9 Consensus mechanism

The Moracle network utilizes the same consensus mechanism as Lisk, in which voted-in delegates take turns forging blocks.[2] Delegates will be voted for specifically on the Moracle sidechain, and will forge transaction and data blocks. However, only one list of ranked delegates for the entire network will exist. The top *n* (an arbitrary number of delegates allowed to forge, in the Lisk mainnet this is set to 101) overall delegates will be allowed to forge transaction blocks in the transaction chain, while the top *n* delegates in any individual data chain will be allowed to forge blocks in their respective data chain. A delegate can be a member of as many data chains as they desire, however they will need to send a data-chain registration transaction (see Section 2.6) in order to do so. This section is subject to change as the details on what DPoS features the Lisk API will offer for sidechain applications.

3. Use cases

Moracle is a fully decentralized and autonomous notary / oracle service. It could be used in many blockchain applications including:

- Applications that on cryptocurrency exchange rates pulled from web APIs
- Decentralized social networks and news sources with no single point of failure.
- Proof-of-identity services, allowing for decentralized permanent records of public encryption keys or other secure information.
- Any decentralized app that needs to harness the power of internet data!

In order to be as useful as possible, many decentralized applications need the ability to get data from the internet without losing the deterministic nature of blockchain technology. A decentralized notary service as described will allow for this.

4. Conclusion

We propose a system to allow decentralized applications to interact with internet or external blockchain resources without trusting a centralized service. Utilizing a main leger to log the existence and location of all data pieces on the network combined with segregated data chains to actually store the information, the service will be able to scale to any size without putting extreme stress on any participating nodes. A delegated proof-of-stake system is utilized for consensus, which saves large amounts of electricity and provides greater reliability when compared to proof-of-work.

References

[1] Filecoin Team. Filecoin <http://filecoin.io/filecoin.pdf>, July 15, 2014
[2] "Consensus." Lisk, The Lisk Foundation, docs.lisk.io/v1.1/docs/the-lisk-protocol-consensus.