PRE-REQUISITE READING

Whitepaper / Terminology

System Overview

BandChain is a high-performance public blockchain that allows anyone to make a request for APIs and services available on the traditional web. It is built on top of the Cosmos SDK ↗, and utilizes Tendermint ↗'s Byzantine Fault Tolerance ↗ consensus algorithm to reach immediate finality. This finality is specifically reached upon getting confirmations from a sufficient number of block validators.

Introduction

While the majority of existing smart contract

arbitrary programs, they lack access to real world data. This limitation hinders the potential of the applications that are developed on those platforms.

BandChain solves this important issue by connecting public blockchains to real-world, off-chain information. The project was created with the following design goals:

1. **Speed and Scalability**: The system must be able to serve a large quantity of data to multiple public blockchains with minimal latency, while maintaining a high throughput. The expected response time must be in the order of seconds.

2. **Cross-Chain Compatibility**: The system must be blockchain-agnostic and able to serve data to most publicly available blockchains. Verification of data authenticity on the target blockchains must be efficient and trustless by nature.

3. **Data Flexibility**: The system must be generic and able to support different methods of retrieiving and aggregating data,

available data as well as information guarded by centralized parties.

BandChain achieves the aforementioned goals with a blockchain specifically built for off-chain data curation. The blockchain supports generic data requests and on-chain aggregation with WebAssembly-powered oracle scripts. Oracle results on BandChain blockchain can be sent across to other blockchains via the Inter-Blockchain Communication protocol (IBC) or through customized one-way bridges with minimal latency.

Terminology

Data Sources

A data source is the most fundamental unit in BandChain's oracle system. It defines the procedure to retrieve raw data from a source and the fee associated with the data query.

On BandChain, a data source can be registered

gh
the registrant sending a `MsgCreateDataSource`

message to the chain. In this message, they specify various parameters the data source they wish to register, including

- the **sender** who wish to create the data source
- the **owner** of the data source, if specified
- the **name** of the data source
- the per-query **fee** that someone looking to use that data source needs to pay
- the **executable** to be run by validators upon receiving a data request for this data source

Examples

The following two examples illustrate what a data source executable might look like. Both examples are written in bash ↗.

Retrieve Cryptocurrency Price from CoinGecko

The data source requires that cURL ↗ and jq ↗ are installed on the executable runner's machine and expects one argument; the currency ticker

```
#!/bin/sh

# Cryptocurrency price endpoint: https://www.coinge
URL="https://api.coingecko.com/api/v3/simple/price
KEY=".$1.usd"

# Performs data fetching and parses the result
curl -s -X GET $URL -H "accept: application/json"
```

Resolve Hostname to IP Addresses

Again, this script assumes that getent ↗ and awk ↗ are available on the host and the host is connected to the DNS network.

```
#!/bin/sh

getent hosts $1 | awk '{ print $1 }'
```

Oracle Scripts

When someone wants to request data from BandChain, however, it is not the data sources that they interact with. Instead, they do so by calling one of the available oracle scripts.

encodes:

- the set of raw data requests to the data sources it needs
- the way to aggregate raw data reports into the final result

Oracle scripts are also Turing-complete and can be programmed in multiple languages. This composability and Turing-completeness makes oracle scripts very similar to smart contracts ↗.

To create an oracle script, the creator must broadcast a `MsgCreateOracleScript` to BandChain. The contents of the message is simlar to `MsgCreateDataSource`, and includes:

- the **sender** who wishes to create the oracle script
- the **owner** of the oracle script, if specified
- the **name** of the oracle script
- the **OWasm** compiled binary attached to this oracle script
- the **schema** detailing the inputs and outputs of this oracle script, as well as the

- the **URL** for the source code of this oracle script

Similar to data sources, the sender who wishes to create the oracle script does not have to be the same as the owner of the oracle script specified in the message.

The execution flow of an oracle script can then be broken down into two phases.

In the first phase, the script outlines the data sources that are required for its execution. It then sends out a request to the chain's validators to retrieve the result from the required data sources. The content of this consists of the data sources' execution steps and the associated parameters.

The second phase then aggregates all of the data reports returned by the validators, with each report containing the values the validator received from the required data sources. The script then proceeds to combine those values into a single final result.

script. BandChain does not enforce any regulations when it comes to the aggregation method used, and entirely leaves that design decision to the creator of the script or any subsequent editors.

Example

The pseudocode below shows an example of an oracle script that returns the current price of a cryptocurrency. The script begins by emitting requests to validators to query the price from three data sources (i.e. the `request` function calls to CoinGecko, CryptoCompare, CoinMarketCap inside `prepare`). Once a sufficient number of validators have reported the prices, the script then aggregates and averages out the reported values results into a single final result (the `aggregate` function).

In this particular oracle script, the aggregation process starts by summing all of the price values returned by the validators across all data sources, as well as the total number of reports returned. It then simply divides the summed

```
# 1st Phase. Emits raw data requests that the orac
def prepare(symbol):
    request(get_px_from_coin_gecko, symbol)
    request(get_px_from_crypto_compare, symbol)
    request(get_px_from_coin_market_cap, symbol)

# 2nd Phase. Aggregates raw data reports into the 
def aggregate(symbol, number_of_reporters):
    data_report_count = 0
    price_sum = 0.0
    for reporter_index in range(number_of_reporters
        for data_source in (
            get_px_from_coin_gecko,
            get_px_from_crypto_compare,
            get_px_from_coin_market_cap,
        ):
            price_sum = receive(reporter_index, da
            data_report_count += 1
    return price_sum / data_report_count
```

Raw Data Reports

Raw data reports are the results that BandChain's validators return when they have successfully responded to a data request and subsequently retrieved results from the required data sources. In these reports, the validators list out the result they got from each data source, using the data source's external ID as the

used to reference a data source within an oracle script, and each data source's external ID is unique within the context of that script.

Oracle Request Proof

When the final data request result is successfully stored onto BandChain, an oracle data proof is produced. This proof is a Merkle proof that shows the existence of the final result of the data request on BandChain. In addition to the actual result value of the request, the proof contains information on the request parameters (oracle script hash, the parameters, the time of execution, etc) as well as as well as those of the associated response (e.g. number of validators that responded to the request). This proof can then be used by smart contracts on other blockchain to verify the existence of the data as well as to decode and retrieve the result stored. Both of these can be done by interacting with our lite client ↗.

Network Participants

network participants, each owning BAND

tokens. In the Laozi mainnet, these participants can be broken down into three main groups; validators, delegators, and data providers.

Validators

Validators are responsible for performing two main functions on the network. First, they are responsible for proposing and committing new blocks to the blockchain. They participate in the block consensus protocol by broadcasting votes which contain cryptographic signatures signed by each validator's private key. This is similar to most other Cosmos-based delegated proof-of-stake blockchains.

Each validator will have a certain amount of BAND tokens bonded to them. The source of these tokens can either be their own holdings, or the tokens delegated to them by other token owners. In most cases, there will be a large number of parties with tokens staked to them. In that case, the top 100 validator candidates with the most token staked to them will become BandChain's validators.

The role the validators described above is similar to those of validators on many other Cosmos-based blockchains. In addition, most transactions supported by BandChain (asset transfer, staking, slashing, etc.) are also derived from Cosmos-SDK.

What makes BandChain unique, and the origin of the validators' second duty, is the chain's capability to natively support external data query. This role will be further explore in the Oracle Data Request Flow section.

Delegators

The second main group of participants are then the individual BAND token holders. On BandChain, BAND holders do not stake their tokens directly, but delegate holdings to a validator. This allows token holders who don't want to set up a validator node to participate in staking rewards.

Data Providers

ird
kind of participant in the BandChain network:

data providers.

With the introduction of on-chain payments, API or data providers can now monetize their data and services directly on BandChain. This new flexibility benefits the network in multiple ways.

Data providers now have a new medium to collect revenue from. As the fees are collected per-query, the revenue that they stand to collect will scale alongside the adoption and usage of BandChain and our oracle as a whole.

The option for data providers to monetize their services directly on-chain will also bring official support for premium and paid data sources onto BandChain. This will allow any developer building on BandChain to access a much wider array of providers and data types they can choose from, enabling BandChain oracle infrastructure to power a much wider range of applications and services.

Oracle Data Request Flow

an be broken down into four main steps:

1. Publishing the necessary data sources and oracle scripts to the network
2. Sending the oracle data request transaction
3. Fetching the necessary data
4. Aggregating and storing the request result onto BandChain

1. Publishing Data Sources and Oracle Scripts

Before any data requests can be made, two conditions must be met:

1. The oracle script that describes the data request must also have been published to Bandchain via `Msg`
2. The data sources related to the aforementioned oracle script must be published to BandChain

2. Oracle Data Request Initialization

Once the required data sources and oracle

a

request to Band's oracle by broadcasting

`MsgRequestData`. The contents of the message includes the ID of the oracle script that the requester wants to invoke and other query and security parameters.

Once the data transaction is confirmed on BandChain, the requested oracle script will begin its execution. The script's execution process can be split into two phases.

3. Fetching the Data

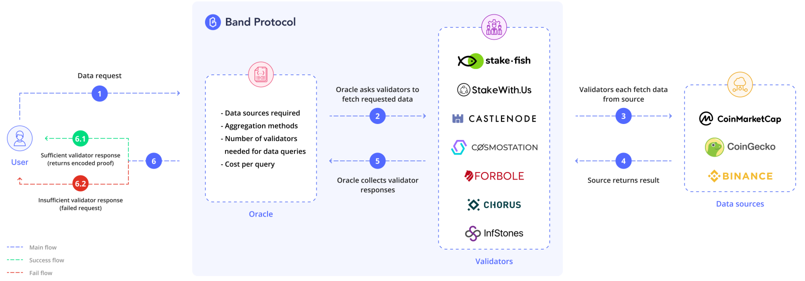
First, the oracle script's preparation function will emit the set of raw data requests necessary to continue the script's execution. The chain's validators, who are chosen at random for security reasons, will then inspect the raw data requests and execute the associated data sources' procedures as instructed by the request. Specifically, each of the chosen validator will attempt to retrieve information from all of the data sources specified in the executed oracle script.

The validators that successfully retrieved data got from each of the data sources, by broadcasting `MsgReportData`. Once a sufficient number of validators, specified in the data request's security parameters, have reported the their results, BandChain will begin executing the oracle script's second part of aggregating request result.

Note that for data from permissioned sources (e.g. under paywall), the data sources are expected to verify that payment has occurred on BandChain and supply data to requested validators accordingly. That way, BandChain allows API providers to monetize data with BandChain's on-chain payment settlement without needing to trust a middleman party.

4. Aggregating and Request Result Storage

This phase begins by aggregating all of the validators' reports (which contains the data each received from the data sources) into a final single result. This final result is then permanently stored in BandChain's application on the chain's state tree and can be sent to other blockchain.



When the final result is successfully stored, an oracle data proof is also produced. This proof is a Merkle proof that shows the existence of the final result of the data request as well as other related information (oracle script hash, the parameters, the time of execution, etc) on BandChain. This proof can then be used by smart contracts on other blockchain to verify the existence of the data as well as to decode and retrieve the result stored. Both of these can be done by interacting with our lite client.

Terminology

Token Economics

BAND PROTOCOL