Apsis: HTTPS-Empowered Blockchain

Shengda Ding

1 Overview

Satoshi Nakamoto's Bitcoin introduces the concept of blockchain and signals the emergence of crypto currency. It is effectively the first blockchain application in the world. Ethereum expands the application scope of blockchain, extending from distributed payment network to generic computation platform for decentralized contracts and applications. Decentralized finance, mostly notably known as DeFi, is the dominant decentralized application on blockchain. Since Compound launched liquidity mining in August 2020, we have witnessed a more than 10 times growth in DeFi total locked value within 6 months.

The prosperity of DeFi exposes several shortcomings of existing blockchains, including low throughput and isolation from the external world. While the former can be addressed by new technologies such as sharding and rollup, the latter has no clean solution yet since existing blockchains are inherently isolated decentralized networks. Currently DeFi applications heavily rely on the so-called **off-chain Oracles**, which is backed by one or more off-chain operators, to help retrieve the external data and feed it back on-chain. Since the performance of the off-chain operators cannot be effectively validated on-chain, off-chain Oracles become the security bottleneck of DeFi applications.

Apsis empowers blockchain applications with HTTPS access capabilities. By introducing retroactive verifiability to HTTPS invocations, Apsis allows smart contract developers to implement **on-chain Oracles** which can invoke HTTPS API synchronously and reliably in blockchain transactions. Apsis also introduces hybrid blockchain applications which can further expand the application scope of blockchains and take full advantage of both Web 2.0 and Web 3.0.

2 Oracle: Solutions and Problems

Since its inception, blockchain has been an isolated and self-contained computation platform. If the data required is not available on chain, it must be fed on chain in order to be processed by smart contracts. Any role that takes the data feeder responsibility is called Oracle.

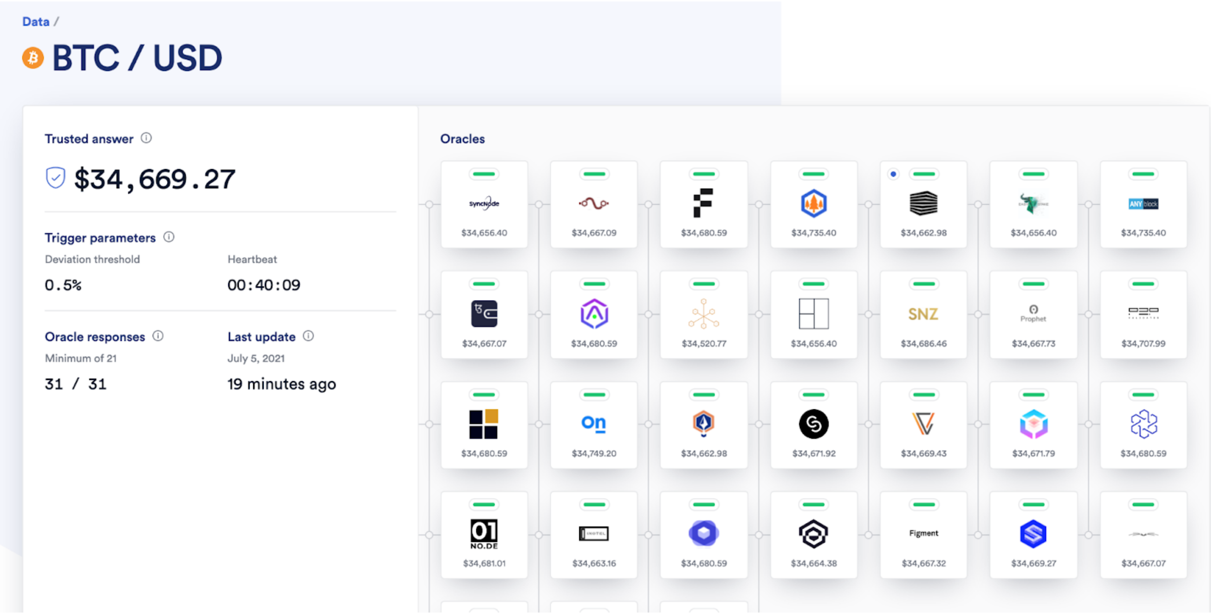
Blockchain applications, especially the emerging DeFi applications, heavily rely on off-chain data provided by Oracles to function properly. For example, lending and exchanges rely on Oracles to provide asset prices, and prediction markets rely on Oracles to provide certain results, e.g. the winner of 2020 US presidential election. However, compared to the blockchain itself, existing Oracle solutions can only offer extremely limited security and reliability guarantees, which makes it the shortest stave in Liebig's barrel for blockchain applications.

2.1 Existing Oracles Solutions

This section goes through several major Oracle solutions in both Ethereum and Polkadot ecosystems. Note that Nest is not discussed here since we are targeting generic Oracle solutions.

2.1.1 ChainLink

ChainLink is undoubtedly the largest and most widely used Oracle in DeFi. ChainLink centrally manages a list of trusted Oracle nodes which are operated by well-known companies. Oracle nodes in the same data market, e.g. BTC/USD price, form a Decentralized Oracle network for that data. Each Oracle node manages their own list of data sources, including free and premium ones, and aggregates the data using their own algorithm.



Data /

BTC / USD

Trusted answer

$34,669.27

Trigger parameters

Deviation threshold: 0.5%

Heartbeat: 00:40:09

Oracle responses (Minimum of 21): 31 / 31

Last update (July 5, 2021): 19 minutes ago

Oracles

$34,656.40 $34,667.09 $34,680.59 $34,735.40 $34,662.98 $34,656.40 $34,735.40

$34,667.07 $34,680.59 $34,520.77 $34,656.40 $34,686.46 $34,667.73 $34,707.99

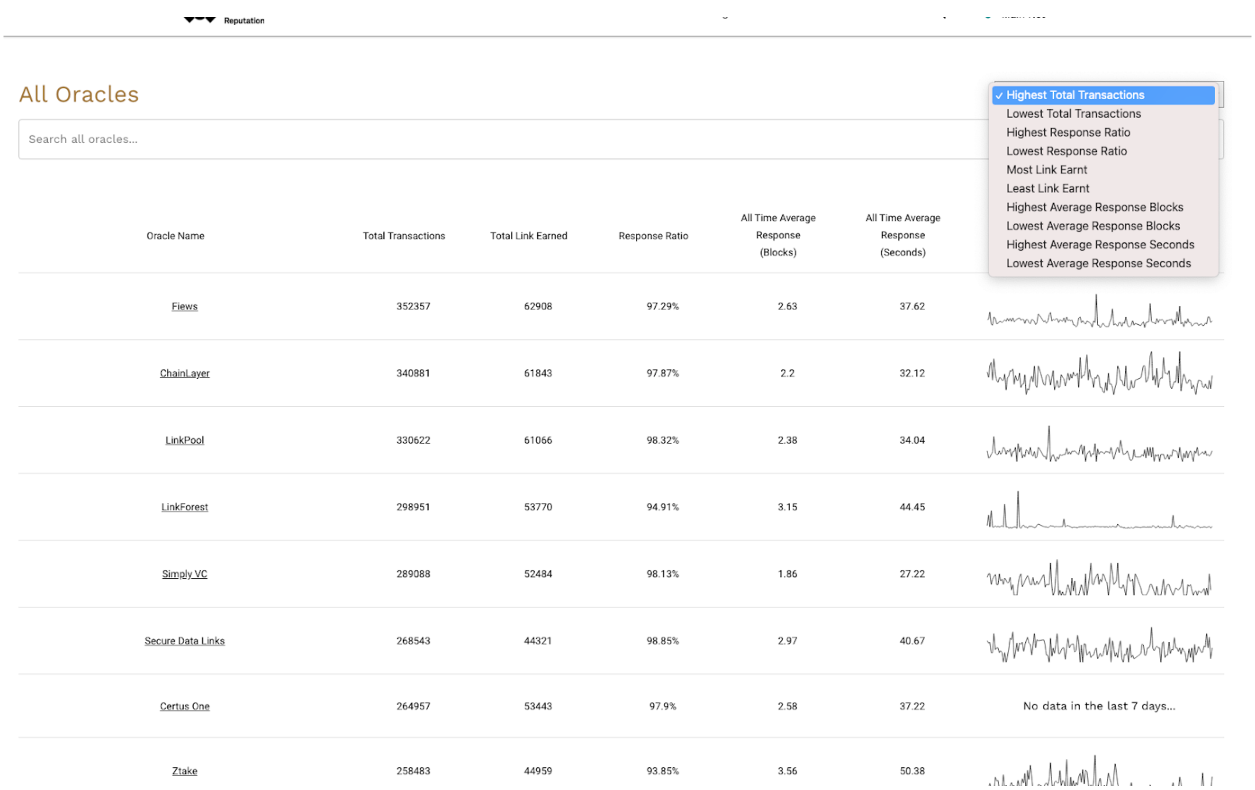
$34,680.59 $34,749.20 $34,662.98 $34,671.92 $34,669.43 $34,671.79 $34,680.59

$34,681.01 $34,663.16 $34,680.59 $34,664.38 $34,667.32 $34,669.27 $34,667.07

When the target price deviates from a certain threshold or the heartbeat timer expires, nodes in the Decentralized Oracle Network report their new data points which are aggregated and reported on-chain by a randomly selected network leader.

ChainLink is effectively a centralized data marketplace. The Oracle nodes which form a decentralized network to aggregate and report price data are centrally managed by the ChainLink team. Moreover, the Oracle node is opaque to public users. Critical information such as the data sources used are hidden so that no one can verify where the data is read from and whether the nodes are reading and aggregating data honestly.

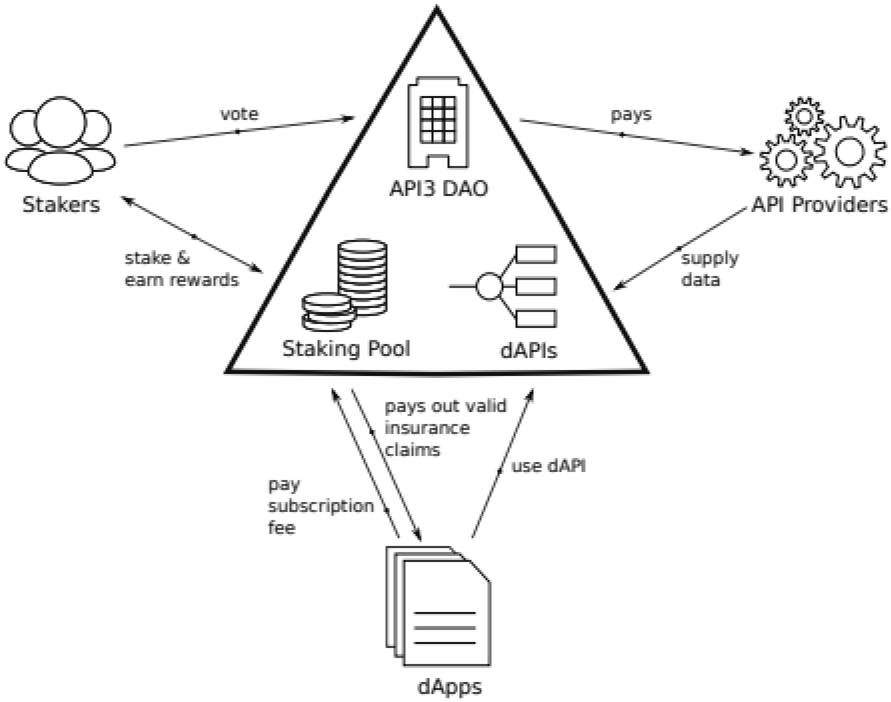
ChainLink attempts to build a reputation system for Oracle nodes, but the current reputation metrics are purely based on Oracle node responsiveness instead of the quality of data reported.


Reputation
All Oracles
Search all oracles...
✓ Highest Total Transactions
Lowest Total Transactions
Highest Response Ratio
Lowest Response Ratio
Most Link Earnt
Least Link Earnt
Highest Average Response Blocks
Lowest Average Response Blocks
Highest Average Response Seconds
Lowest Average Response Seconds

Oracle Name	Total Transactions	Total Link Earned	Response Ratio	All Time Average Response (Blocks)	All Time Average Response (Seconds)	
Fiews	352357	62908	97.29%	2.63	37.62	
ChainLayer	340881	61843	97.87%	2.2	32.12	
LinkPool	330622	61066	98.32%	2.38	34.04	
LinkForest	298951	53770	94.91%	3.15	44.45	
Simply VC	289088	52484	98.13%	1.86	27.22	
Secure Data Links	268543	44321	98.85%	2.97	40.67	
Certus One	264957	53443	97.9%	2.58	37.22	No data in the last 7 days...
Ztake	258483	44959	93.85%	3.56	50.38	

2.1.2 API3

API3, which is designed and implemented by a former ChainLink Oracle node operator, aims to solve the data opacity issues in ChainLink. They find that the data providers, which often have big off-chain reputations, are completely hidden in the ChainLink architecture. Therefore, instead of letting Oracle nodes earn the middlemen tax, they introduce Airnode, a first-party Oracle node which is operated by the data providers directly.



API3 makes the data providers visible to data consumer DApps. However, similar to ChainLink, public users still cannot verify whether the data really comes from the target data providers.

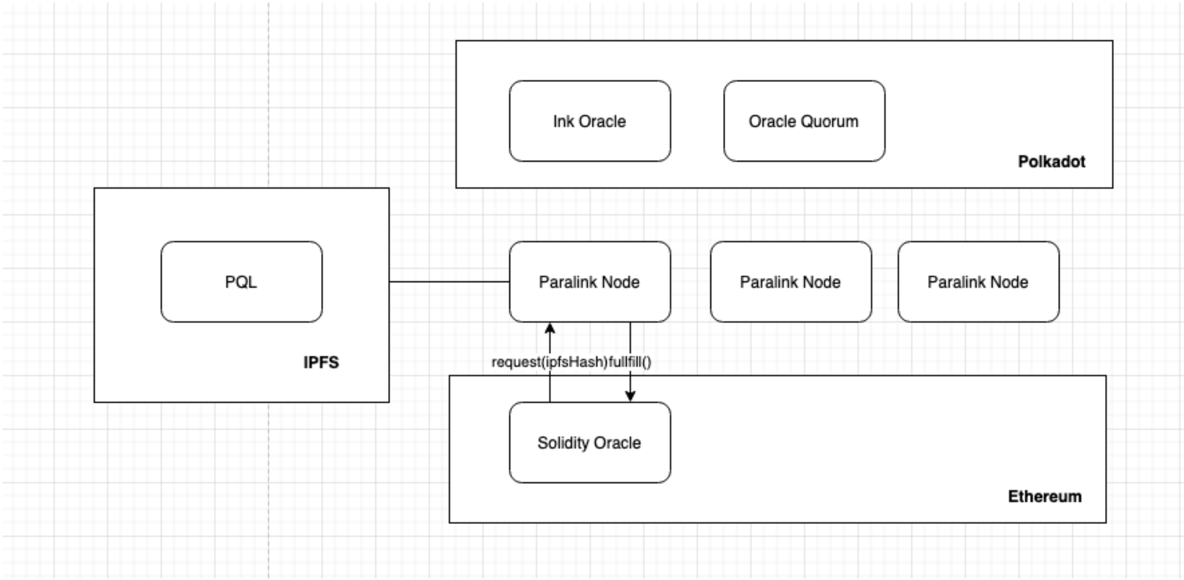
API3 is also difficult to use. In order to access a new data provider, DApp developers must request API3 DAO to create a new decentralized API, or dAPI for short, for that data provider. After dAPI is established, DApp developers then should subscribe to the dAPI, create a subscriber address and fund it with sufficient ETH so that the data providers can use it to supply data.

In short, API3 is a custom solution targeted at the data providers, but it does not solve the data opacity issue in Oracle and instead introduces new usability issues to data consumers.

2.1.3 Paralink

Paralink is an Oracle solution on Ethereum and Polkadot. Similar to ChainLink and API3, Paralink utilizes off-chain Oracle nodes, called Paralink nodes, to access external data. However, Paralink offers two additional improvements:

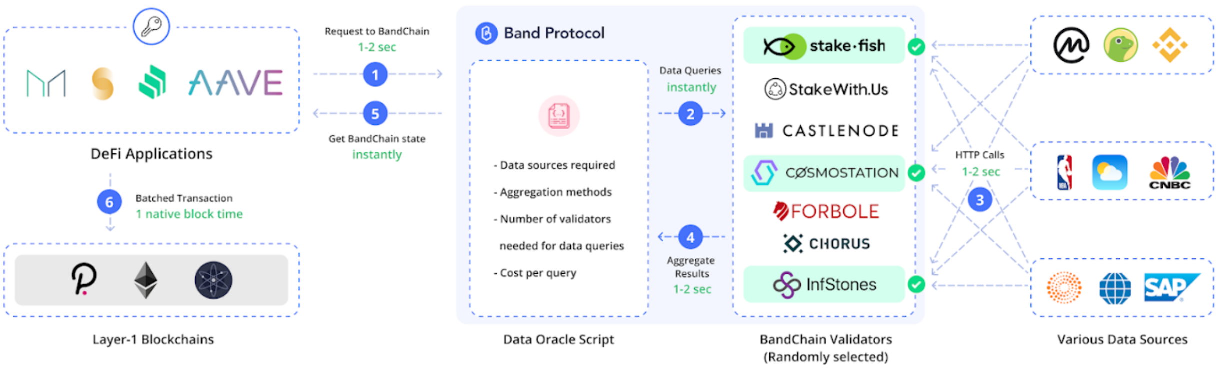
1. Paralink defines an Oracle script language, named Paralink Query Language, to enable flexible data source definition and data aggregation operations;
2. Paralink offers on-chain Oracle nodes quorum on Polkadot so that the Oracle nodes are managed decentralized.



However, similar to ChainLink and API3, Paralink Oracle users need to trust Paralink Node to execute their Oracle script honestly. Public users cannot verify the outcome of Oracle scripts.

2.1.4 Band Protocol

Band Protocol builds a Cosmos-based blockchain, called Band Chain, to perform Oracle operations to cross-chain data consumers.



Similar to Paralink, Band Chain defines an Oracle script which allows Oracle users to specify the data source, data aggregation and data validation operations. For each data source, Band Chain uses DPoS algorithm to randomly select multiple validators to read data from the data source. Data read by the validators are aggregated and their medium is reported as the final result to data consumers.

Band Chain is one step ahead of the other Oracle solutions discussed above as it creates a dedicated Oracle blockchain. This allows public users to verify the past output of the Oracle from the chain. For example, if an Ethereum transaction should consume price data from the

Band Chain, public users are able to verify how the data is aggregated from various data points. However, users cannot verify whether the data points come from the actual target data sources.

There is also no guarantee that Band Chain validators do their right job since a medium aggregation approach cannot preclude collusion. For example, says Band Chain allocates 3 validators for each data source, if the attackers happen to control 2 of the select validators, they will be able to forge the data reported from this data source.

2.2 Problems of Off-chain Oracles

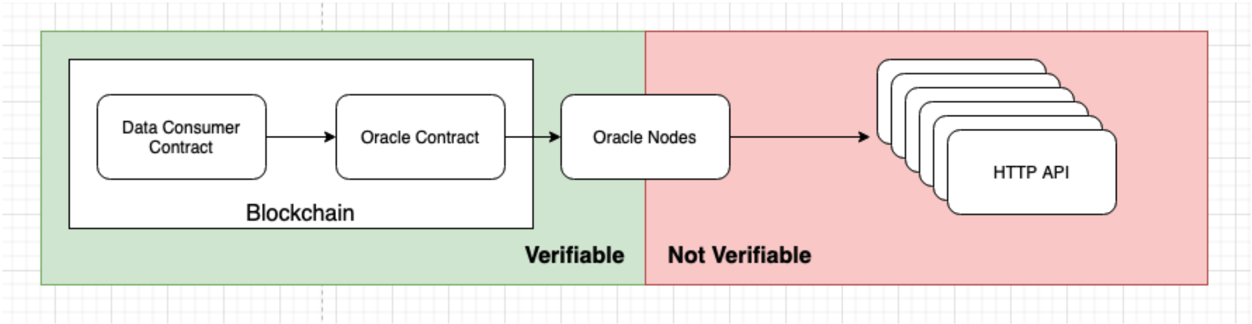
The Oracle solutions discussed above can all be classified as off-chain Oracles, since they have off-chain segments on the data path between data providers and data consumer DApp. Therefore, they share similar drawbacks as discussed below.

2.2.1 No Off-chain Verifiability

The most critical issue of off-chain Oracles is the lack of data verifiability on their off-chain segment. Therefore, data consumers have no way to validate whether the data consumed is generated from the target data source.

The lack of off-chain verifiability applies to all existing Oracle solutions:

- In ChainLink, data sources are hidden by Oracle nodes so that Oracle data consumers have no idea where the data comes from;
- In API3 and Paralink, even though data sources are visible, Oracle users have to trust the Oracle nodes to read data from data source honestly;
- In Band Protocol, a quorum of validators read from the same data source. However, the medium approach cannot preclude collusion and it does not work for all data types.



As we could see from above, the data path of off-chain Oracles can be divided into two segments: The segment between DApp and Oracle nodes, and the segment between Oracle nodes and data sources(HTTP API). While existing Oracles strive to secure the first segment, the second segment remains unverifiable which makes the whole data path unverifiable. Note that we cannot simply re-invoke the data source to verify the data provided, since the data read might change already in the data source. This is especially true for historical transactions since they are minted long ago.

Therefore, with the lack of off-chain verifiability, Oracle data consumers have to trust Oracle in order to consume their data.

2.2.2 Overreliance on Token Economy

Since off-chain data cannot be verified on-chain, existing Oracles rely on token economics to incentivize Oracle node operators to perform honestly. Token economy works for Bitcoin and Ethereum to bootstrap their security foundation but does not work for Oracles.

For Bitcoin and Ethereum, if any miner attempts to forge a transaction or a block, the forged transaction or block will be excluded by honest miners right away because they can be effectively validated. Oracle, however, is more of a hindsight: the dishonest action can only be recognized only after certain real damage is done because the Oracle data cannot be validated at the time of posting. In order to incentivize honest operators, the on-going Oracle rewards should always be higher than the one-off income of being dishonest.

However, real life situations are always more complicated. As suggested in API3' whitepaper:

1. People tend to undervalue their future income of being honest, and overvalue the one-off income of being dishonest due to the uncertainty of future;
2. The one-off income of being dishonest can be huge, considering the burst of DeFi total locked value. For example, since Oracle price data is used to determine whether a lending position should be liquidated or not, Oracle operators might choose to forge the data and grab the liquidation benefit.

Therefore, off-chain Oracles have to overpay their operators, which makes the Oracle management extremely expensive and unsustainable. The Oracle might still fail if their token price drops, or new huge dishonest rewards appear. Moreover, overpayment does not preclude the free riders. Oracle node operators can simply forge price data by returning a medium of historical prices over a period of time, instead of reading price data from premium data sources. Free riders have low risks of being discovered even though low-quality data is provided.

2.2.3 Pull vs Push Methods

Off-chain Oracles offer two approaches for DApp to access external data: the pull and push methods.

- In the pull method, DApp triggers a data request event signaling the desired data to retrieve. Oracle nodes pick up the request, complete the reading and aggregation job, and send data back to DApp;
- In the push method, data is pushed to the Oracle contract by Oracle nodes periodically or when a certain threshold is met.

The pros and cons of each method are discussed in the table below.

	Pros	Cons	Example
Pull method	Data is up-to-date; support dynamic request URL	Asynchronous invocation, difficult for programing	ChainLink v1, Band Chain, API3, Paralink
Push method	Synchronous invocation, programming friendly	Data is stale; difficult to set proper push schedule; data URL must be known beforehand	Band Protocol v1, ChainLink v2

Even though the pull method tends to offer fresher data than the push method, the freshness is still dependent on the Oracle nodes. If the Oracle node does not pick up the request and complete the job in time, pull-based Oracle users still suffer from outdated data.

3 Apsis Consensus

We believe that off-chain Oracle is a compromise and temporary solution to access external data. If a blockchain itself can access HTTP API persistently and reliably, blockchain applications can read external data in their transaction instead of trusting an external off-chain party to do that on their behalf. Apsis aims to become the first HTTPS-empowered blockchain which can take full advantage of the security of Web 3.0 and the diversity of Web 2.0.

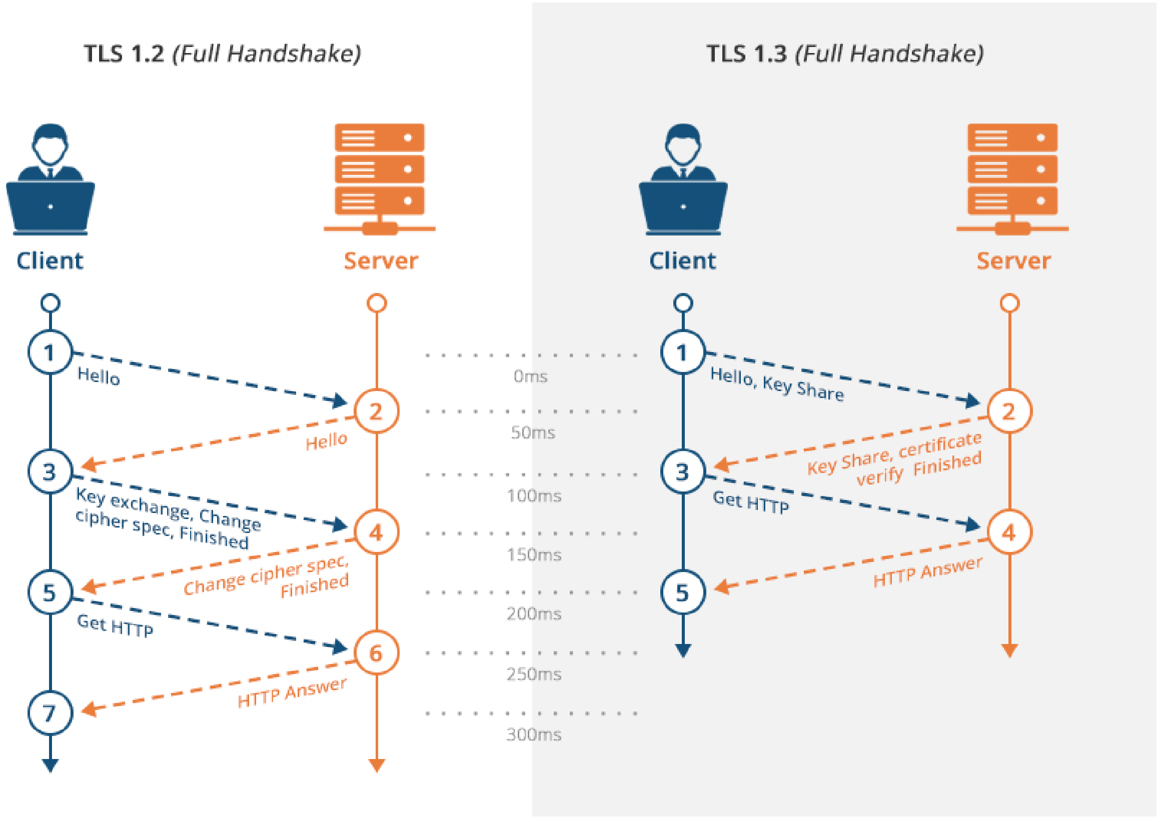
3.1 Retroactively Verifiable HTTP Invocations

Blockchain transaction data needs to be retroactively verifiable. The validity of past transactions can be verified by any others, even when these transactions are mined on-chain a long time before. HTTP, on the other hand, does not provide such native retroactive verifiability. We cannot verify an HTTP response by simply re-invoking the HTTP API because the HTTP API data might change already, This is especially true for HTTP responses recorded long ago.

HTTP invocation can be retroactively verified as long as the following assumptions are met:

1. HTTP server is using HTTPS with TLS 1.3;
2. HTTP server has a valid SSL certificate;
3. HTTP server returns a valid Date header.

Even though TLS 1.2 can offer similar verifiability, we are dedicated to TLS 1.3 due to its forward security, efficiency and simplicity. Below is a comparison between the TLS 1.2 and TLS 1.3 handshake process.



If all three assumptions are met, we are able to construct a proof for an HTTP API invocation with the following elements:

1. HTTPS server certificate
2. Diffie-Hellman parameters
3. HTTPS server signatures on Diffie-Hellman parameters
4. Encrypted HTTP request and response

HTTPS server certificate certifies the HTTPS server public key. Combined with Diffie-Hellman parameters and the server signatures used in the HTTPS session, we are able to generate and verify the share key used in this session, which is in turn used to verify the HTTP request and response content.

In other words, the first two assumptions, i.e. HTTPS with valid certificates, allow us to generate a proof that the data is read from a target server honestly. This makes the HTTP API invocation verifiable. The third assumption, i.e. a valid Date header in the HTTP response, makes this verification retroactive with a persistent proof for the HTTP response timing. All these combined generate a proof of whether and when an HTTP API is invoked so that it can work with other on-chain transaction data.

One might argue that not all Web 2.0 HTTP servers can meet these three assumptions. We will discuss how we handle SSL certificates in section 3.2. For assumption 1 and 3, fortunately, almost all HTTP servers nowadays work with HTTPS and they are able to return a standard valid Date header. For those incompatible servers, they will be precluded implicitly by the blockchain applications. Since blockchain applications require gas to deploy smart contracts and execute transactions on-chain, they work for high-stake applications with high security and reliability demand. Therefore, it’s highly unusual if they select an API that does not support HTTPS or does not follow the HTTP standard.

In short , Apsis chain is not supposed to support all Web 2.0 servers, but instead works with HTTP servers that are needed by blockchain applications.

3.2 SSL Certificate Integration

As mentioned in section 3.1, the second assumption to meet is that HTTPS server should have a valid SSL certificate. To verify an HTTPS invocation record, Apsis miners need to validate its SSL certificate at the same time of the HTTPS invocation, since this SSL certificate might become invalid at the time of validation.

To help achieve global consensus on SSL certificate validity, Apsis integrates SSL certificate validation as part of the blockchain implementation. Apsis maintains both the root CA certificates and the Certificate Revocation Lists(CRL) on-chain. Whenever new certificates are revoked, Apsis DAO, which is discussed in detail in Section 3.4, submits a change to CRL so that the newly revoked certificates, along with the revocation date, are updated on-chain.

The process to validate an SSL certificate at a specific timestamp is as follows:

1. Check whether the certificate has valid proof from CA. If no, the certificate is invalid;
2. Check whether the certificate expires before the target timestamp. If yes, the certificate is invalid;
3. Check whether the certificate is on the CRL. If so, check whether its revocation timestamp is before the target timestamp. If yes, the certificate is invalid;
4. Otherwise, the certificate is valid at the target timestamp.

Since anyone can validate an SSL certificate for any given timestamp, Apsis chain can be seen as a public welfare chain on its own, since it could solve the availability and usability issues of the current Online Certificate Status Protocol(OCSP).

3.3 APS and Apsis DAO

APS is the native token of the Apsis chain. Users need to consume APS as the gas fee to deploy smart contracts and execute transactions. Similar to ETH, APS is used to cover the computation and storage cost at Apsis chain, especially the additional cost of HTTPS invocation and HTTPS session proof storage.

APS holders can stake APS to become, or delegate others to become validators of the Apsis chain. Apsis chain adopts Delegated Proof-of-Stake (DPoS) as its underlying consensus protocol. Proof-of-Work(PoW) is definitely not an option since Apsis transactions might interact with HTTP servers. If PoW is adopted, multiple miners might be executing the same transaction with HTTP invocation, which effectively becomes a DDoS attack to the HTTP server.

APS stakers, no matter whether they delegate themselves or others, will become members of Apsis DAO to participate in Apsis chain governance. Apsis DAO is responsible for the following works:

- Empower Apsis chain development;
- Bootstrap and incentivize Apsis ecosystem building;
- Perform routine chain governance operations, such as update the latest Certificate Revocation List on-chain.

4 Apsis EVM

Apsis is an HTTPS-empowered blockchain which supports retroactively verifiable HTTPS invocations. The application scope of Apsis can be maximized with Apsis EVM, an HTTPS-empowered smart contract platform which allows smart contracts to invoke HTTPS API synchronously and reliably.

The HTTPS access capability in Apsis EVM is provided with an EVM precompile. Below is a sample of the HTTP precompile written in Solidity.

```
contract HTTPClient {
  // Format of the HTTP response content.
  enum Format{ JSON, TEXT, BINARY };

  // Reads data from an HTTP endpoint.
  // @param url URL to invoke
  // @param format Format of the HTTP response content
  // @param path Path of the data to return
  // @param expiration Deadline of the HTTP invocation
  // @return The data specified as JSON path
  function getData(string url, Format format, string path, uint256
expiration) public returns (bytes memory);
}
```

The following code snippet shows how to invoke HTTPS API inside a solidity method.

```
function rebase() external {
  const _httpClient = HTTPClient.at(‘0x0000000000008’);
  const _data = httpClient.getData(‘https://api.token.com/prices/AMPL’,
HTTPClient.Format.JSON, ‘price.value’, 10 minutes);

```

```
  uint256 _price = abi.decode(_data, {uint256});
  _rebaseWithPrice(_price);
}
```

This code snippet tries to read the latest price of AMPL token to determine whether a rebase operation is needed. Assume that the (faked) URL https://api.token.com/prices/AMPL returns a JSON which contains AMPL's latest price under path 'price.value'. When the getData method is invoked on the HTTPClient precompile, it invokes https://api.token.com/prices/AMPL, reads and parses the JSON response, and returns the bytes data as result. Miners will append proof of the HTTPS invocation, as specified in section 3.1, as part of the transaction log.

Apsis EVM provides a much enhanced development experience than off-chain Oracles.

- Compared to pull-based Oracles, Apsis smart contract can invoke HTTPS API synchronously as part of the transaction. Smart contract logic is thus significantly more concise;
- Compared to push-based Oracles, Apsis smart contract supports ad-hoc HTTPS API URL which is infeasible with push-based Oracles;
- Apsis smart contract can access fresher data then both pull-based and push-based Oracles since data is read in the same transaction.

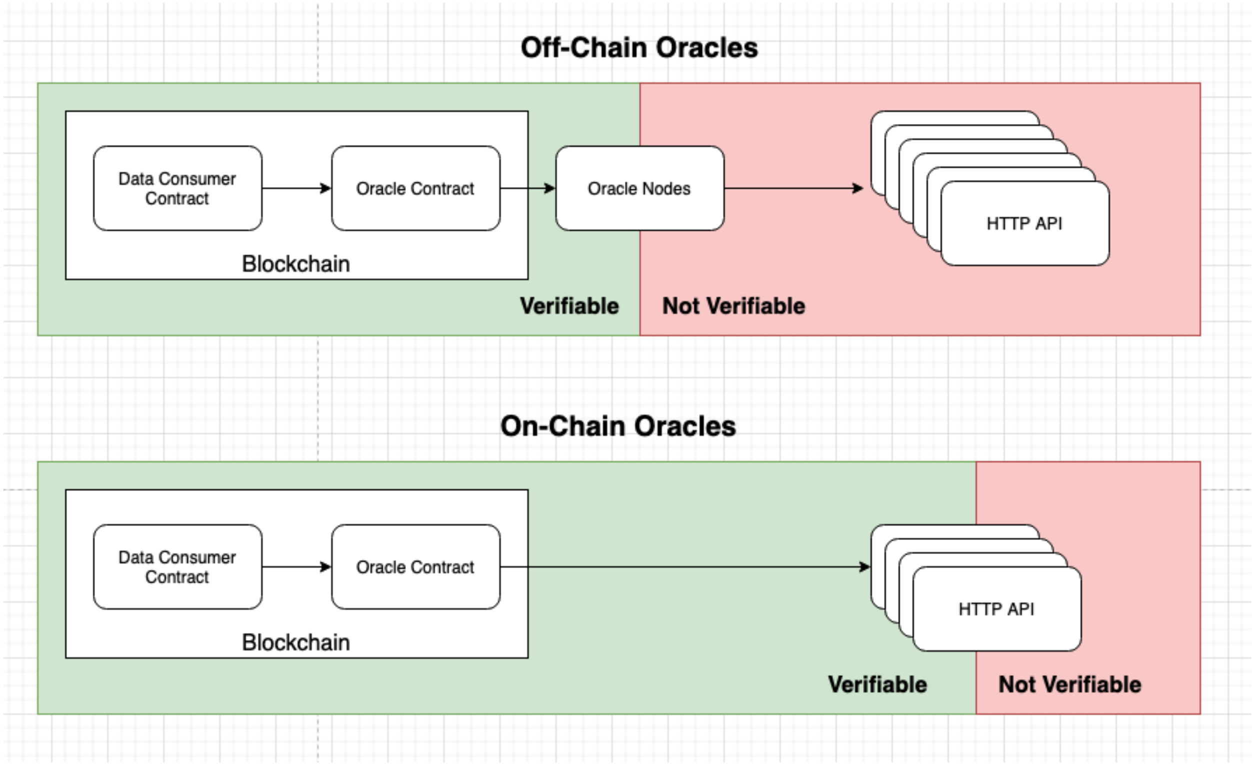
One potential problem is the storage size. As transactions involving HTTP invocation accumulate, HTTPS session proof is likely to bloat and consume large amounts of storage space. This problem can be addressed by offloading HTTPS session proofs to external decentralized storage. For example, for relatively old blocks, we could package HTTPS session proofs for every 100 blocks and store them in other decentralized, content-addressable storage solutions such as IPFS, while leaving the response data(e.g. the return value of getData() in the example above) stored on-chain. Therefore, if miners want to validate relatively old transactions, they can download the HTTPS session data and discard them afterwards.

5 Apsis Applications

Blockchain application scope can be significantly expanded with HTTPS access capabilities. This section lists several potential applications of the Apsis chain.

5.1 On-Chain Oracles

One notable application of Apsis is On-chain Oracles which provides end-to-end verifiability to access external data.



There are two critical differences between off-chain Oracles and on-chain Oracles:

- Oracle nodes are no longer required in on-chain Oracles since on-chain Oracles can access HTTP API directly. This might imply significant operational cost since no middleman tax is paid to Oracle nodes;
- Data from HTTP API to Oracle contract is completely verifiable in on-chain Oracles. This means the data consumers only need to trust the HTTP API providers, in contrast to off-chain Oracles where data consumers need to trust the Oracle nodes to operate honestly.

The presence of on-chain Oracle does not preclude the existence of third-party Oracle providers. Existing Oracle providers can build their Oracle solutions on Apsis which read data from multiple HTTP APIs and aggregate data points. There are several benefits to build on-chain Oracle solutions on Apsis:

- On-chain Oracles on Apsis provide end-to-end data verifiability. Oracle data consumers don't have to trust the Oracle providers as the whole data generation process is visible on-chain;
- Oracle providers don't have to manage their Oracle node networks and overpay them with their own tokens. Instead, Oracle providers can better utilize their token to bootstrap the ecosystem;
- On-chain Oracles can support both realtime and cached data access. Oracle data consumers can pay more to trigger a new round of data update in order to read the latest data, or pay a small portion to access the data cached in the last round update.

On-chain Oracles can also be accessed by applications on other blockchains via cross-chain bridge. For example, lending protocols on Ethereum can utilize the Apsis-Ethereum bridge to read the latest price from the on-chain Oracles on Apsis. Apsis chain can become an Oracle hub in the incoming multi-chain era.

5.2 Web 2.0 Enhancement

Apsis chain is also an enhancement to Web 2.0. Since blockchain applications are expensive to execute, they are mostly high stake applications with high security and reliability demand. It is not a surprise if they expect the same security and reliability on their dependent HTTP APIs.

In order to track the performance of HTTP APIs, traditional Web applications need to build their own tracing solutions, recording individual responses from the API providers, and providing proof if they believe their API providers do not perform as expected.

With Apsis, all responses from API providers are visible on-chain and can be verified by any public users. If any DApp incidents happen, it is not difficult to determine whether the issue comes from the dependent HTTP APIs or the blockchain application itself. For example, if one's lending position is liquidated due to low loan-to-value ratio, we could figure out whether it's caused by asset price change or smart contract flaw. If it's caused by asset price change, say, a drop in collateral price, we are able to figure out how the price is read and calculated, eventually finding out what data sources drive the price shift. The whole liquidation process is transparent.

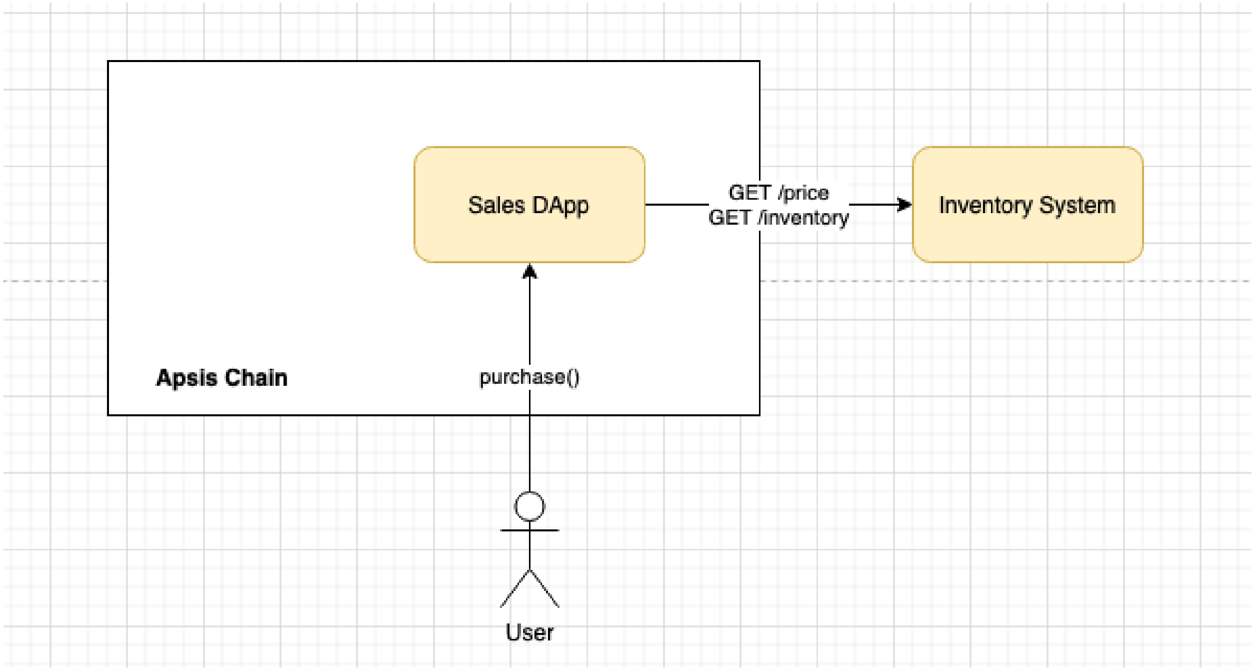
In short, Apsis brings retroactive verifiability to Web 2.0 APIs. One could build a reputation system based on the HTTP API provider performance since their past responses are recorded on-chain and can be verified by any users.

5.3 Hybrid Blockchain Applications

The true potential of the Apsis chain is definitely not limited to on-chain Oracles. Since smart contracts on Apsis can access HTTP API synchronously within a single transaction, developers are able to write smart contracts involving complicated HTTP interactions, which are currently not available in any existing blockchain.

In other words, Apsis opens the space for totally new kinds of applications, which we call hybrid blockchain applications, to take full advantage of both the security and reliability from Web 3.0, and diversity and complexity from Web 2.0.

Assume that an electronics retailer wants to build a blockchain application to sell their televisions. Since there are many TVs available and their inventory information changes frequently, the retailer does not want to maintain all information on-chain. Instead, they rely on their in-house inventory system to provide price and inventory information. Users can purchase TV with cryptocurrency via the sales blockchain application and anyone can verify the sale transaction information on-chain.



It is difficult to implement the sales DApp using existing Oracle solutions. Push-based Oracle doesn't work since it will be expensive to push all prices and inventory information on the chain. Pull-based Oracle also has its own issues.

- First, the purchase experience will have to break into multiple transactions since the pull-based Oracle is asynchronous. The problem is worse as the purchase involves multiple HTTP interactions, especially when these HTTP interactions depend on each other. In such a case, the users might have to sign multiple transactions with long time waiting to complete their purchase;
- Second, both users and the retailer have to trust the Oracle to return the correct information from the inventory system. If something goes here, say, a TV is sold at $10, it would be difficult to figure out where the mistakes come from.

All these problems can be solved with Apsis. Since Apsis enables smart contract transactions with synchronous HTTPS invocations, the sale DApp can complete the purchase experience in a single transaction. In addition, if a TV is sold at $10, we could easily identify whether the $10 price comes from the inventory system since all HTTPS interactions are recorded and can be verified by any users. Moreover, by eliminating the Oracle nodes as the middleman, the development and operational cost can be substantially smaller.

6 Security Analysis

The end-to-end security of Apsis transactions depends on both Apsis chain itself and the HTTP server. If we treat the HTTP server as a blackbox, we can show that:

1. When the HTTP server is secure, no one can forge transactions or invalidate a valid translation on Apsis;
2. When the HTTP server is breached, Apsis can provide a security guarantee no worse than the existing Oracle solutions.

The former is easy to prove. If one is able to forge an Apsis transaction which consists of at least one HTTPS invocation, they need to forge an SSL session shared key. To do that, they have to either obtain the HTTPS server private key or force the HTTPS server to use their forged SSL certificates. This requires breaking the HTTPS server or the encryption algorithm. Therefore, Apsis can offer much higher security and reliability guarantee than existing Oracle solutions if the HTTPS server is secure.

Even though the security of HTTP server is external to Apsis' security model, Apsis is no worse than the existing Oracle solutions when the HTTPS server is breached. We will discuss the detailed cases in the following sections.

6.1 HTTP Server Certificate is Breached

If the SSL certificate of the HTTP server is breached, the attacker can forge an Apsis transaction by creating a fake HTTPS response as proof. Likewise, in the existing Oracle solutions, the attacker can easily launch a man-in-the-middle attack against Oracle nodes. In a word, the HTTP server is no longer secure for either Apsis or Oracles.

Assume that the certificate breach is unknown to HTTP server operators. Neither Apsis nor Oracles can detect the breach, but Apsis can facilitate the discovery since the forged HTTPS sessions are recorded on-chain which allows HTTP server operators to verify the past sessions.

Assume that the certificate breach is known and thus added to RCL. How Oracle detects revoked SSL certificates is implementation specific. On the other hand, since Apsis is governed and operated by Apsis DAO, any Apsis DAO members can propose an RCL update on the chain. The decentralized nature of Apsis can reduce the damage time caused by the server certificate breach.

6.2 HTTP Server DNS is Hijacked

When the HTTPS server DNS is hijacked, the attackers still need to create a falsified SSL certificate to launch an attack. This is similar to what is discussed in section 4.1, and Apsis can help to detect falsified SSL certificates and react promptly.

6.3 HTTP Server is Breached

If the HTTP server itself is breached, neither Apsis miners nor Oracle nodes could do anything about it since the attacker can forge any response. However, Apsis can facilitate the breach discovery and investigation process. With all API responses recorded on-chain, users can identify the data coming from the HTTP server and locate the root causes in the HTTP server.

6.4 HTTP Server Returns Different Contents

HTTP servers might return different contents to different callers, even when they are invoking the same URL at the same time. For example, an HTTP response might be customized based on the caller's location or language. Even though it causes confusion to Apsis miners, we don’t consider it as error cases due to the following reasons:

1. If the HTTP server does not return a deterministic result, then this HTTP server might not be a good candidate for blockchain applications which requires high security and reliability;
2. Oracle might be able to alleviate the issues by aggregating results from multiple nodes, but it is not a generic approach and it can only provide an estimate for the data.

Therefore, if one Apsis miner receives a valid HTTP response and uses it to construct a valid block, it is considered as a valid transaction execution process.