PYTH NETWORK: A FIRST-PARTY FINANCIAL ORACLE

PYTH DATA ASSOCIATION

VERSION 2.0

September 28, 2023

Financial market data is often only accessible to a limited set of institutions and users. Traditional markets typically maintain strict control over live and historical price feeds. Cryptocurrency markets currently have fewer barriers, although there is no guarantee this arrangement will continue. Consequently, only a selected group of users has access to the most timely, accurate, and valuable information.

The Pyth network aims to bring this valuable financial market data to DeFi applications and the general public. The network does so by incentivizing market participants — trading firms, market makers, and exchanges — to share the price data collected as part of their existing operations. The network aggregates this first-party price data and publishes it on-chain for use by either on- or off-chain applications.

The original whitepaper proposed an initial protocol design focused on the Solana blockchain. This second version updates the design to reflect the current status of the Pyth ecosystem. In particular, this version describes a novel cross-chain model for price delivery to target chains.

The Pyth Data Association (the "Association"), in collaboration with the community of Pyth network participants, has published this whitepaper to describe the current state of the Pyth network.

1 OVERVIEW

The Pyth protocol is designed to incentivize participants to continually publish price updates for various products (such as BTC/USD). Each product has a price feed that continually updates with its current price and a confidence interval representing the estimated uncertainty of the price. For example, the current BTC/USD feed may say the price is $\$65000 \pm \$50$. The feed for each product is published on-chain, where it is read by *consumers*, who may be either blockchain-enabled programs (on various blockchains) or off-chain applications. An on-chain program produces the price feed for each product by aggregating the price feeds of individual *publishers*. The protocol is designed to attract publishers who are first-party data providers with the ability to source high-quality, timely pricing information.

Thus, the protocol has two different sets of participants:

- **Publishers** publish price feeds and earn fees in exchange. Publishers are typically market participants with access to accurate and timely price information.
- **Consumers** read price feeds and incorporate data into smart contracts or dApps. In the process, consumers pay fees to the protocol to access the data.

Fees are charged at a fixed rate per Pyth price update on a target chain – e.g., 0.0001 ETH per update. Consumers are users of on-chain protocols that integrate Pyth for market data. These protocols may be running on various blockchains.

To make Pyth prices available to a wide range of consumers, the protocol must operate over multiple blockchains. The Pyth protocol uses an appchain called Pythnet to store and update the state of each price feed. Pythnet is a proof-of-authority blockchain where each publisher runs a validator. Pyth prices are broadcast from this appchain to other *target chains* by way of a cross-chain architecture that uses decentralized cross-chain messaging protocols, such as the Wormhole network. Broadcasting data from Pythnet makes it easy for the protocol to support a large number of target chains, thereby enabling consumers to use Pyth data no matter what blockchain they are on.

Within this architecture, the aforementioned participants will interact via two mechanisms. Both of these mechanisms will be implemented on one of the various chains detailed above:

- **Price aggregation** combines the price feeds of individual publishers into a single price feed for the product. This mechanism is designed to produce robust price feeds, that is, feeds whose prices cannot be significantly influenced by small groups of publishers. This mechanism runs on Pythnet as an on-chain program.
- **Pyth Governance** determines high-level parameters of the other mechanisms.

A critical challenge is designing these mechanics to be robust to various forms of price manipulation. Two specific attacks to consider are:

1. Participants could onboard as publishers and attempt to manipulate the oracle price. The price aggregation mechanism is designed to guard against this attack by limiting the influence of publishers on the aggregate price. Furthermore, the community will determine how publishers are permitted to use the protocol and can choose to limit the set to highly reputable and honest participants.
2. Bad actors could try to move prices on exchanges to influence the aggregate Pyth price. Robustness in the price aggregation and a diversity of publishers sourcing from different exchanges can decrease the risk of price manipulation efforts.

2 CROSS-CHAIN ARCHITECTURE

Pyth features publishers who provide price data on Pythnet, where the aggregation mechanism (described in the next section), produces aggregate prices. The aggregated prices are subsequently broadcast and transferred to other blockchains where consumers can use them. Pythnet is an application-specific blockchain on which the only programs and activity are related to price publishing, aggregation, and initiation of the transfer to other blockchains. Pythnet's consensus model and runtime are based off of those of Solana.

A successful cross-chain architecture must achieve several important properties:

1. **Minimize transaction costs**. Maintaining price feeds on multiple blockchains requires paying transaction costs (i.e., gas) on every single chain. The resulting cost could be too expensive for the participants to bear.
2. **Support high-frequency and low-latency price updates**. These two properties are essential for applications that require timely price information.

3. **Scale to many blockchains**. There are use cases for prices on different blockchains that feature a diversity of runtimes, cosnensus models, and gas costs. If cross-chain transfer of price data is too expensive, it will not be possible to reach all these chains.

4. **Scale to many price feeds**. The Pyth protocol should scale to support price feeds for a wide range of crypto and non-crypto assets. As with scaling to many blockchains, cross-chain transfer of price data being too expensive will inhibit reaching this scale.

The cross-chain architecture is a "pull" oracle that addresses these challenges using a three-part workflow. The first part of the workflow is publishing prices on chain efficiently and cheaply. In contrast to generalized blockchains that can feature high gas costs, on Pythnet, permissioned publishers are able to publish their updates for free. These permissioned publishers can submit price updates with a valid price and confidence to the on-chain oracle program, which stores their values in their field of the corresponding on-chain price account. When invoked, the oracle program carries out aggregation on the prices in the price account (see Section 3 for more details).

The second part is a price broadcast operation performed by the Pythnet validators as part of Pythnet consensus. After the oracle program aggregates publishers' prices into an aggregate price for each of the price feeds updated in a slot, it calls into the message buffer program with the aggregate prices of the updated price feeds. The message buffer program allows for the creation of a message to be sent to other chains and can be called by authorized programs. Here, the message buffer stores the aggregate prices of the updated price feeds. Next, the validator reads the message buffer's accounts and constructs a merkle tree with the aforementioned prices as the leaves of the tree. It then broadcasts the root of the tree to other chains using a cross-chain messaging protocol, such as the Wormhole network. The Wormhole network is a decentralized cross-chain messaging system that enables data on one blockchain to be read on other chains. The broadcast operation is intended to be invoked at a high frequency, essentially at the frequency of the Pythnet blocktime. The output of this broadcast is a stream of attested merkle tree roots that are available via the cross-chain messaging infrastructure and the data of the trees themselves publicly visible on Pythnet for a rolling window of historical slots.

The third part is relaying, which occurs on a target chain where Pyth prices are available. Each target chain has a Pyth receiver contract which stores a price for every Pyth price feed. These contracts implement a permissionless operation to update the stored price. The input to this operation is an attested merkle tree root, a price update for a particular symbol that represents a leaf in the corresponding merkle tree, and the path in the tree that leads from that leaf to the root. The latter two can be accessed publicly on Pythnet. The operation performs various checks on the message (e.g., to validate signatures, to validate the root-leaf-path combination); if the update is valid, the new prices are stored in the Pyth receiver contract.

Consumers of Pyth prices are responsible for invoking the permissionless price update, i.e. pulling the price update on chain, before using the price. The price stored in the Pyth receiver contract grows stale over time unless it is updated. Protocols integrating with Pyth price feeds can impose a limit on the maximum staleness of the price. Anyone interacting with one of these protocols will therefore be required to invoke the permissionless price update before their interaction if the current on-chain price is stale.

In this design, consumers pay the transaction costs for updating Pyth price feeds *only when the price is needed*. This property is how the cross-chain architecture meets the goals set forth above. The broadcast component can stream high-frequency price updates for many different price feeds without incurring any transaction costs. Many of these price updates will never land on a target chain (and therefore never incur a transaction cost). Target chain transaction costs are themselves minimized because consumers only incur them when they need to use a price for an operation.

2.1 Update Fees

Consumers are required to pay an *update fee* to the protocol in order to invoke the price update operation on a Pyth receiver contract. Fees are charged in the native token of the target chain.

2.2 Comparison: Pull vs. Push

Most oracles have traditionally used a push architecture. In this architecture, the oracle publishes prices onto a general-purpose blockchain, and may even perform aggregation on that chain. The oracle is responsible for keeping prices fresh, and therefore must regularly send transactions to update the on-chain price. These transactions incur gas costs that scale with update frequency and the number of supported price feeds, blockchains, and possibly even publishers. Moreover, these oracles can be unreliable, as their price update transactions compete with other network activity. During busy periods, network congestion can prevent oracle updates from landing on-chain.

Unlike the push model, Pyth's pull model features price publishing and aggregation on a cheap chain. Pythnet is a fork of Solana, which allows for low-latency and high-frequency price updates. Pythnet transactions are free for publishers, which significantly lowers the overall operating costs of the oracle. Broadcasting prices to other blockchains still carries gas costs. However, unlike the push model where updates are pushed (and gas is paid) irrespective of demand, Pyth only pays gas when consumers demand a price update. As a result, gas consumption is more targeted and efficient. Consequently, Pyth can have more frequent price updates, and scale to more price feeds and blockchains than push oracles. The pull model is also not prone to unreliability. Network congestion on target chains does not affect the ability of publishers to publish prices to Pythnet, and motivated users can always pull a price from Pythnet on to their target chain by paying a sufficient transaction fee.

Another advantage of Pyth's architecture is its ability to handle stateful computations like exponentially-weighted moving averages (EMAs). These computations are time-weighted combinations of previously published data. Push oracles struggle to compute such combinations on-chain, as their low update frequency produces coarse temporal samples. For instance, it is difficult to construct a 5-minute EMA when the update frequency is 10 minutes. In contrast, Pyth has access to price updates every 400 ms on Pythnet, which easily allows it to construct EMAs and other temporal combinations.

3 PRICE AGGREGATION

The price aggregation mechanism combines each individual publisher's price and confidence feed into a single aggregate price and confidence feed. For example, one publisher may say that the price of BTC/USD is $\$52000 \pm 10$ and another that it is $\$53000 \pm 20$, and price aggregation may combine these two prices into an aggregate price of $\$52500 \pm 500$. This mechanism is part of the on-chain program and triggers when publishers submit price updates: the first price update on a given slot automatically aggregates the prices from the previous slot.

The price aggregation algorithm is designed to have three properties:

1. It is resistant to manipulation — both accidental and intentional — by publishers. For example, if most publishers submit a price of $100 and one publisher

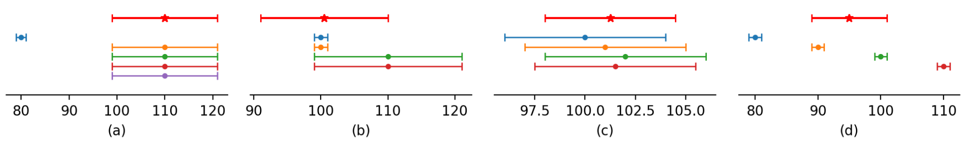

Figure 1: Scenarios for the aggregation procedure. The lower thin bars represent each publisher's prices and confidence intervals, and the bold red bar represents the resulting aggregate price and confidence.

submits a price of $80, the aggregate price remains near $100. This property increases the likelihood that the aggregate price remains accurate even if a small percentage of publishers submit a price far from the market. Figure 1(a) depicts this scenario.

2. The aggregate price appropriately weights data sources with different levels of accuracy. The Pyth protocol allows publishers to submit a confidence interval because they have varying levels of accuracy in observing the price of a product. For example, some publishers are expected to be exchanges. Exchanges have different levels of liquidity, and less liquid exchanges tend to have wider bid/offer spreads than more liquid ones. This property can result in situations where one exchange reports $\$101 \pm 1$, and another reports $\$110 \pm 10$. In these cases, the aggregate price is closer to $101 than $110. Figure 1(b) depicts this scenario.

3. The aggregate confidence interval reflects the variation between publishers' prices. In real-world markets, there is no single price for any given product. Products trade at slightly different prices at various venues at any given time. Furthermore, the product's spread is a fundamental limit on the precision of the product's price. The aggregate confidence interval reflects both the variation across venues and these limitations. Figures 1(c) and (d) depict two different cases where there are price variations across exchanges.

The price aggregation algorithm uses a variant of the weighted median. An input to the algorithm is a stake-weight for each publisher. These weights will initially be set uniformly, but in the future may be set using additional mechanisms (such as data staking from the original whitepaper) as determined by the community. The first step of the algorithm computes the aggregate price by giving each publisher three votes – one vote at their price and one vote at each of their price plus and minus their confidence interval – then taking the stake-weighted median of the votes. The second step computes the distance from the aggregate price to the stake-weighted 25th and 75th percentiles of the votes, then selects the larger as the aggregate confidence interval.

This simple algorithm is a generalization of the ordinary median. Most people understand the median as the middle value in the data set, that is, the 50th percentile. However, the median is also the value R that minimizes the objective function $\sum_i |R - p_i|$ where p_i is the price of the ith publisher. This function penalizes R based on its distance from the publisher's price p_i. The proposed algorithm computes the aggregate price R that minimizes $\frac{1}{3}\Sigma_i s_i |R - p_i| + \frac{2}{3}\Sigma_i s_i \max(|R - p_i| - c_i, 0)$, where s_i is the publisher's stake-weight and c_i is the publisher's confidence interval. This objective does two different things. First, it weights publishers according to their stake, such that low-stake publishers have minimal influence on the price. Second, it combines the ordinary median objective with a second term that only assigns a penalty to R if it lies outside the publisher's confidence interval.

4 GOVERNANCE

On-chain governance is expected to be responsible for taking the following actions:

- Determining the size of update fees.
- Determining the reward distribution mechanism for publishers.
- Approving other software updates to on-chain programs across blockchains.
- Determining how products are listed on Pyth and their reference data (e.g., number of decimal places in the price, reference exchanges).
- Determining how publishers will be permissioned to provide data for price feeds

5 TOKEN DISTRIBUTION

	Unlocked	Locked	Total
Publisher Rewards	0.5	21.5	22
Ecosystem Growth	7	45	52
Protocol Development	1.5	8.5	10
Community and Launch	6	0	6
Private Sales	0	10	10
Total	15	85	100

Table 1: Allocation of locked and unlocked PYTH tokens. Locked tokens unlock 6, 18, 30 and 42 months after token launch.

There are 10,000,000,000 PYTH tokens and this total supply will not increase. Furthermore, 85% of the tokens are initially contractually locked. Locked tokens unlock 6, 18, 30, and 42 months after token launch. This schedule is designed to produce a gradual increase the unlocked token supply over time. The remaining 15% of PYTH tokens are initially unlocked. The supply of both locked and unlocked tokens are allocated per the categories shown in 1.

6 CONCLUSION

This whitepaper proposes an oracle protocol to make accurate, high-resolution financial market data easily accessible on-chain. The protocol is designed to be a self-sustaining decentralized network that coordinates data publishers and consumers. The protocol is designed to attract publishers with high-quality pricing data, and incentivizes these publishers by paying them fees collected from consumer usage of the data. The mechanisms help prevent malicious attackers from manipulating the protocol to their benefit in various ways, such as manipulating the price.