Autopilot

Overview

The autopilot is the engine that drives forward CoW Protocol.

Architecture

Running at a regular interval, it keeps an up-to-date view on the state of the protocol, synthesizes this data into anauction , broadcasts this auction to each of the solvers, and finally chooses which solution will be executed.

Users don't interact with the autopilot directly: its only intended interface is with the solvers. There is a single autopilot running for each chain1 .

Its role can be broadly summarized into these main purposes.

1. Cutting auctions
2.
    - Data availability consensus around which orders are valid for a given batch
3.
    - The initial exchange rates for each traded tokens which are used to normalize surplus across different orders
4. Solver competition
5.
    - Data availability consensus around the available solution candidates and their score
6.
    - Notifying the winning solver to submit their solution
7. Auction data storage

Methodology

Cutting auctions

The autopilot builds an auction that includes all tradable orders. To handle this, it needs to maintain a complete overview of the protocol.

Much of the order data it needs is collected through the database, which is shared with the orderbook. The database stores the vast majority of available order information, including user signatures.

Other information can only be retrieved on-chain and is updated every time a new block is mined. For example, it needs to know from the protocol:

- Whichpre-signatures
- have been set
- If newnative token orders
- have been created
- Tracking which orders have beeninvalidated
- by the user
- Detecting if a batch has been settled and it should prepare a new auction

Retrieved information isn't limited to the CoW Protocol itself. The autopilot needs to provide a reference price for each token in an order (a numéraire); the reference price is used to normalize the value of thesurplus , since the surplus must be comparable for all orders and two orders could use the most disparateERC-20 tokens. The reference token is usually the chain's native token, since it's the token used to pay for the gas needed when executing a transaction. Orders whose price can't be fetched are discarded and won't be included in an auction.

Native token price fetching is handled by an integrated price estimator in the autopilot. The price is fetched from multiple sources and may change based on the current configurations. Prices are both queried from a list of selected existing solvers as well as retrieved internally by the autopilot (for example, by querying some external parties like Paraswap and 1inch, but also by reading on-chain pool data as Uniswap).

Orders that can't be settled are filtered out. This is the case if, for example:

- an order is expired
- for fill-or-kill orders the user's balance isn't enough to settle the order
- the approval to the vault relayer is missing
- the involved tokens aren't supported by the protocol

The autopilot also checks thatERC-1271 signatures are currently valid.

More in general, the autopilot aims to remove from the auction all orders that have no chance to be settled. Still, this doesn't mean that all orders that appear in the auction can be settled: orders whose ability to be settled is ambiguous or unclear are remitted to the solvers' own judgment.

Solver competition

Once an auction is ready, the autopilot sends a/solve request to each solver. Solvers have a short amount of time (seconds) to come up with asolution and return itsscore to the autopilot, which represents the quality of a solution. The scoring process is described in detail in thedescription of CoW Protocol's optimization problem . The autopilot selects the winner according to the highest score once the allotted time expires or all solvers have returned their batch proposal.

Up to this point, the autopilot only knows the score and not the full solution that achieves that score. The autopilot then asks the winning solver to reveal its score (through/reveal) and then to execute the corresponding settlement transaction (/settle). The solver is responsible for executing the transaction on-chain (through thedriver if using the reference implementation).

Auction data storage

The data returned by the solver is stored by the autopilot in the database. Other auction data is recorded as well, for example surplus fee for limit orders and the score returned by each solver. It also records the result of executing the settlement on-chain in order to track the difference in score caused by negative or positive slippage.

This data will be used to compute thesolver payouts .

Considerations

Complexities

A typical challenge in the autopilot is handling blockreorgs . The autopilot must be able to revert as many actions as possible in case of a reorg; everything that can't be reverted must be accounted for in the stored data.

In practice this means that some information (e.g., competition data by transaction hash) is only available after a "reorg safe" threshold of blocks have been proposed.

What the autopilot doesn't do

The autopilot doesn't verify that a solver's transaction is valid, nor that it matches the score provided by the solver. For this purpose, it's only responsible for documenting the proposed solution and the effects of a settlement to the on-chain state. Misbehavior is detected and accounted for when computing the solver payouts based on the data collected by the autopilot. The solver payouts are handled outside of the autopilot code.

In the same way, the autopilot doesn't verify that therules of the game have been upheld. This is handled in the solver payout stage as well; in exceptional circumstances the DAO can decide to slash the amount the solver staked for vouching.

Footnotes

1. There is technically also a second autopilot running on each network for testing purposes (in thebarn
2. environment), but this isn't necessary for running the protocol.↩ Edit this page Previous Order book Next Driver