From users, solvers as well as potential business partners we continue to receive the question: Wen L2 Sers?

. In this document, I'm hoping to outline some of the challenges that I see with deploying CoW Protocol on L2, as well as offer a path to how we can reduce the core protocol's complexity to make it easy to scale to new chains assuming sufficient solvers signal support for it.

Why is deploying on L2s complex?

CoW Protocol has come a long way since its launch in early 2021. What started as a simple Swap UI (forked from Uniswap) combined with a few simple "solvers" based on DEX Aggregator APIs and the main value proposition of matching people in batches while taking on the difficulty of transaction management (including MEV protection) has become a fully fledged decentralized exchange protocol. There are now 9 external solvers competing for order flow and being rewarded via the protocol with a sophisticated rewards scheme.

Through time the development team added support for swapping native ETH, placing long lived limit orders, trading with smart contract wallets and creating advanced order types such as TWAP and Stop Loss orders.

The off-chain infrastructure supporting this, cow services, has grown significantly since then and supporting the protocol even just on mainnet is taking a substantial amount of resources. To name a few main areas of effort:

- Reliable, fast and affordable node infrastructure to index baseline liquidity, provide basic price estimates, simulations and balance/signature checks
- Computing accurate slippage accounting and solver reward payouts as well as harvesting fees to convert to ETH week over week, while automated, requires manual verification and its logic has to be frequently adjusted for basically every new feature.
- Providing good and reliable price estimates for users at scale is incredibly important to score high on price comparators such as DefiLlama and amongst pro users. Yet, it has become an increasingly costly and complex part of the backend API to provide those reliably.
- The API uses a heuristic to identify unsupported tokens which have both false positives (leading to reasonable tokens not being tradable) and false negatives (leading to orders that can never be fulfilled). In fact, we see a lot of user feedback that their favorite tokens are not supported on CoW Protocol.
- Providing firm fee estimates for swap orders at the time of order placement exposes the protocol to inventory risk and potential games being played during times of high gas volatility (people waiting for a "favorable" quote).
- In order to save gas, solvers are allowed to "internalize" AMM interactions using the protocol internal buffers (that are filled from collecting trading fees) to settle small trades very cost efficiently. This "rule" introduces complexity and monitoring overhead to make sure the protocol buffers are not exposed to one sided inventory risk.
- Solution submission strategies (ie. using public mem-pool vs MEVBlocker, how much to tip, how and when to cancel a transaction) are challenging on Ethereum alone and will be even harder to do optimally on other chains with different MEV landscapes.

At the same time, decentralizing the solver competition has helped a lot in reducing complexity from the core services, and making sure users always get best execution using the most up to date liquidity sources without requiring work from the core development team.

It is clear that any L2 needs to be bootstrapped with external solver support from the get go. Already on Gnosis Chain, where solver competition is still light, the development team is receiving a lot of requests to add liquidity to the Gnosis solvers, which we struggle to prioritize.

Additionally, we believe that there is still a lot more non-trivial work that the backend services are currently doing, which should be stripped away from the "core protocol" before it gets replicated on other chains.

How to reduce complexity

I will now outline some initiatives that are already being worked on or - in my opinion - need to be completed, before deploying the protocol on a wide range of layer 2 chains becomes feasible.

Solver-Driver Colocation

This initiative has been in the works for a long time now, and will finally be pushed over the finish line in the next few weeks. It will allow solvers to manage the lifecycle of their settlement transaction themselves. Up until now, solvers only provide the "recipe" (ie calldata) for the settlement transaction and the protocol's "driver" assembles, signs and submits the raw transaction data on behalf of the solver.

With solver-driver colocation, solution submission strategies can be fully owned by the solver teams and the protocol is only used as a synchronization point to get consensus on which solver reported the highest objective value and thus won the right to submit their settlement. While large scale deployment on mainnet may require revisiting the current bonding pool structure and potentially more infrastructure to “circuit break” misbehaving solvers, I believe co-located driver/solvers should be a requirement for L2 deployments from day one.

Solver provided Price Estimates

The recently passed DAO proposal creates an incentive mechanism for solvers to not only provide solutions to auctions but also create price estimates (aka quotes) for users that are about to place an order. Up until now this task was done by the backend itself using mainly DEX aggregator APIs and some of the most basic Gnosis solvers. For more background on why this change is beneficial please read this forum post.

As a next step, we’d also like to move “bad token detection”, i.e. the decision which tokens are supported or not by the settlement contract, into price estimation logic. Solvers will provide calldata with their quotes, allowing to simulate the settlement on chain. Even for tokens that the protocol deems unsupported right now (e.g. the ones taking a fee on transfer), If a solver provides a quote simulation that succeeds, the API should accept the order and assume the same solver will also be able to provide a solution for this order later in the auction.

Letting solvers specify fees for all orders

Currently, the protocol has three different types of fee mechanisms:

1. For swap orders, the API estimates a fee that is guaranteed to the user (regardless of gas price fluctuations) assuming their order has a reasonable chance of filling soon.

2. For fill-or-kill limit orders, the backend computes a reasonable fee that will be taken from the user’s surplus and updates it in (almost) each runloop of the auction.

3. For partially fillable limit orders, solvers compute a reasonable surplus fee at the time they provide their solution (since the fee largely depends on the filled amount).

These different methods evolved over time, but it has become clear that solver-specified fees (3.) are the most flexible and exact way of charging fees. Not only do they account for gas savings from CoWs and last minute changes in execution paths or gas price, they also make it very flexible to add a protocol fee that is dependent on execution quality (more on this in the section below).

In any case, maintaining all three modes of fee estimation creates code complexity that we should reduce. Option 3) allows us to support all order types with one fee mechanism. Concretely, this means that 1) style orders would disappear (placing a swap order on CoW Swap would create a limit order with 0 fixed-fee and instead “bake” the fee into the slippage tolerance).

Requiring solvers to capture fees and slippage in their own “Trade Handler”

The current objective criterion maximizes total surplus in ETH + (order fees in ETH - gas cost of execution). The latter brackets part of the equation captures “economic viability” of a solution and basically avoids paying $10 more in gas in order to achieve a $5 price improvement.

Fees are currently accrued in whatever token the order is selling, but could theoretically be sent along and thus “captured” in any token along the trade path (most trades route via ETH at some point).

While our studies show that the inventory risk of holding random sell tokens so far has mostly been negligible, making sure we properly account for fees, slippage and other imbalances and also convert those balances into the native token for gas reimbursements on a weekly basis has been a complex and time consuming task.

Dropping the second term from the equation, while also removing the gas reimbursement for solvers and instead letting them “keep” the order fees (which they are free to route into any token they’d like) would reduce the protocols inventory risk, get rid of the “fee harvesting” process and also remove any games that can be played around “internalizing complex routes”.

Ideally, we would upgrade the protocol to introduce separate settlement contracts per solver (which all have the same bytecode and some shared state such as settled order IDs, but have separate owners which can access the internal buffers).

Even without a smart contract update, this could be achieved by requiring solvers to capture slippage and fees into their own “Trade Handler” contracts (which is something that some solver teams already do). While gas-inefficient on L1, this overhead could be less of an issue on L2.

To summarize, the goal of the “core protocol” should simply be to gather trade intents from users, “cut” auctions and thus creating consensus on which orders are valid to be solved and relevant for the surplus maximizing objective as well as provide a synchronization point and “referee” for the solver competition to collect solutions and figure out which solver should submit their settlement on chain.

With this feature set, complexity will be drastically reduced and allow for CoW Protocol to be easily replicated on other chains.

Transitioning to an L2 tailored revenue model

Deploying the protocol on L2 will inevitably increase the maintenance overhead for the development team.

At the same time, CoW protocol’s current revenue stream, which in the long term needs to sustain the development, today largely comes from “gas savings” that batching multiple orders into a single transaction or “internalizing small trades” creates. We already see that revenue is lower when gas prices are low and observe that revenue on low-fee side chains such as Gnosis Chain is negligible.

In order to align incentives with users and business partners to deploy on L2 we should therefore think about ways how the protocol can generate revenue based on volume, surplus, or positive quote deviation rather than gas improvements (and apply this to mainnet as well).

Having solver provided fees, would allow for very flexible and elegant fee models where e.g. the protocol could after the fact compute what was e.g. the surplus or price improvement compared to the quote the user received and require the solver to pay the protocol a fraction of that value on a regular basis.

Internal analysis showed that e.g. taking 50% of the positive deviations between quote and final execution price (for market orders, all surplus for from limit orders) would have created more than $2.5M in protocol revenue over the last year on mainnet alone.