Bebop JAM: An RFQ Based Meta-DEX Aggregator

June 2023

Mo Nokhbeh
mo.nokhbeh@bebop.xyz

Katia Banina
katia.banina@bebop.xyz

ABSTRACT

We present a DEX aggregator system that allows the exchange of Ethereum based tokens using a Request-for-Quote (RFQ) based approach. First, we describe the current limitations in existing DEX aggregators and recently popular meta DEX aggregators (MDEX) that attempt to create a holistic market. Then we describe JAM as a system of trading that quotes and executes any given trade with no slippage, using solutions determined through competition between solvers. Finally, we conclude how this system addresses the limitations of existing solutions by providing a better price and user experience as well as clearer incentives for all participants.

1. INTRODUCTION

1.1 The rise of DEX

As the issuance of custom tokens became prevalent on Ethereum, decentralised exchanges (DEX) rose to allow the trade of these assets natively on-chain as an alternative to forgoing custody to centralised exchanges.

There was gradual innovation in the DEX landscape, spanning from peer-to-peer orderbooks, to RFQ and subsequently automated market makers. Yet there are continuous improvements on these models as developers attempt to configure the trade-offs resulting in new protocols and upgrades.

1.2 Fragmented Liquidity

As a result of the constant innovation, there were 13 DEXs on Ethereum with over $1m 24h volume as of June 2023 [1]. This scenario created an opportunity for aggregation protocols, allowing users to navigate a fragmented market, and find their path to the best price. Yet, fragmentation wasn't fully solved as some DEXs and aggregators include special arrangements with market makers for providing liquidity. Additionally, for practical reasons, aggregators don't have integrations with all on-chain liquidity and their implementations can produce different outcomes. As a result, some siloes and fragmentation still remain today.

1.3 Meta DEX Aggregators (MDEX)

To further address the fragmented liquidity, meta DEX aggregators have become increasingly popular. They attempt to aggregate from the highest level possible, bringing together aggregators and any potential private liquidity.

In addition to whole market aggregation, it offloads the responsibility to multiple parties (division of labour) allowing to source any liquidity to fulfil user requirements even if those sources are not on-chain or available publicly.

Additionally, the distribution of these trade requests among participants in a meta DEX aggregator creates open competition. This results in a better experience in terms of price, speed and other parameters for users and includes liquidity that may otherwise not have been tapped into.

2. EXISTING SOLUTIONS

In current MDEX solutions there are 3 actors:

- Users: The origin of trade requests for which the system is responsible in fulfilling.
- Solvers: Automated participants that compete to find solutions to the user's trade request, that maximise desirable characteristics such as price and speed.
- Orchestrator: Commonly a centralised service that accepts user trade requests and aggregates

solutions from solvers, ultimately presenting the user with the best one.

To explore the characteristics of the existing solutions we look at the following facets:

1. Price: The method to determine limit prices set for the user ensuring they receive a reasonable result from their trade
2. Competition: The competition between solvers creating solutions for the user's order and what incentives that creates.
3. Execution: The method to execute said solutions on-chain, its reliability and cost.

2.1 CowSwap (Gnosis Protocol V2)

With CowSwap, users are given a limit price that their quote will execute. This limit price is estimated by CowSwap via an aggregation of popular on-chain pools and encoded into the order. CowSwapl queries solvers, comparing their solutions to a batch of orders and then executing the best. Batches are used to create an environment where coincidence of wants (COWs) *(Fig 1)* occur [2]. The theory being that crossing user wants internally in the batch creates a better outcome. In reality, CoWs are a rare occurrence with current levels of liquidity entrusted to the protocol.

Solvers are rewarded in proportion to the improvement over the limit price they provide to the user, hence they are incentivised to maximise this. This reward is handled manually and comes weekly in the form of $COW - CowSwap's governance token. Additionally, in order to cover gas fees that the CowSwap system incurs by handling on-chain execution, solvers are required to reduce their quote output by this given fee amount, for which they are refunded in ETH in the weekly rewards cycle.

2.2 1inch Fusion

1inch followed with their own model. User orders placed with Fusion are published publicly. Solvers make private solutions to the order flow and are responsible for their execution on-chain.

The execution price is encoded in the order on creation and follows a curve where the price declines as block time passes, effectively creating an on-chain auction. The price curve *(Fig 2)* incentivises solvers to submit the transaction on-chain the moment the limit price reaches market.

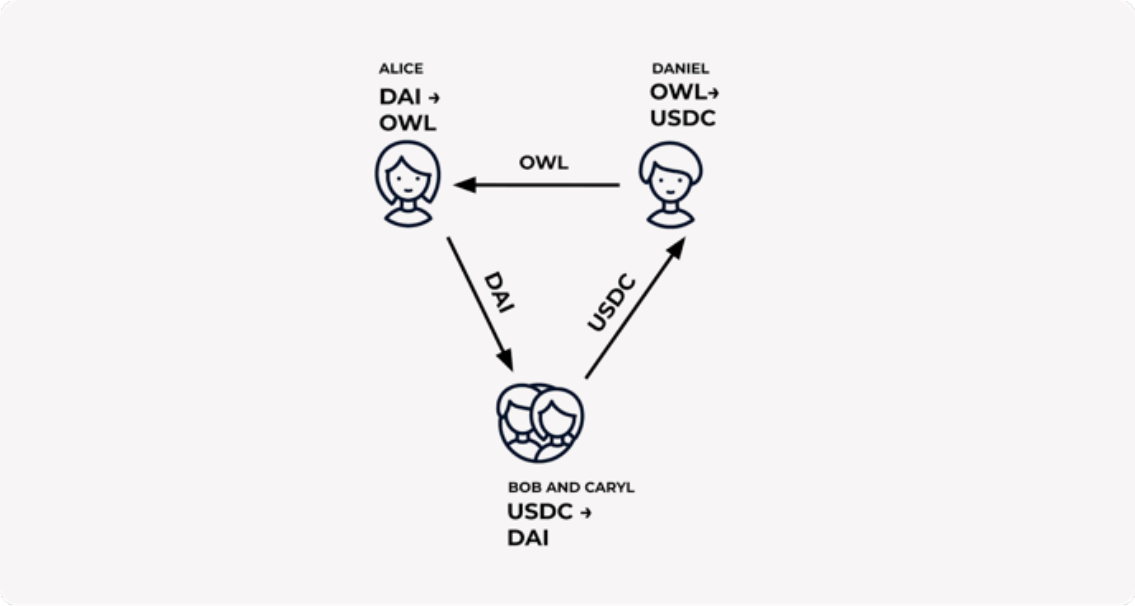

Figure 1.. CowSwap Coincidence of Wants

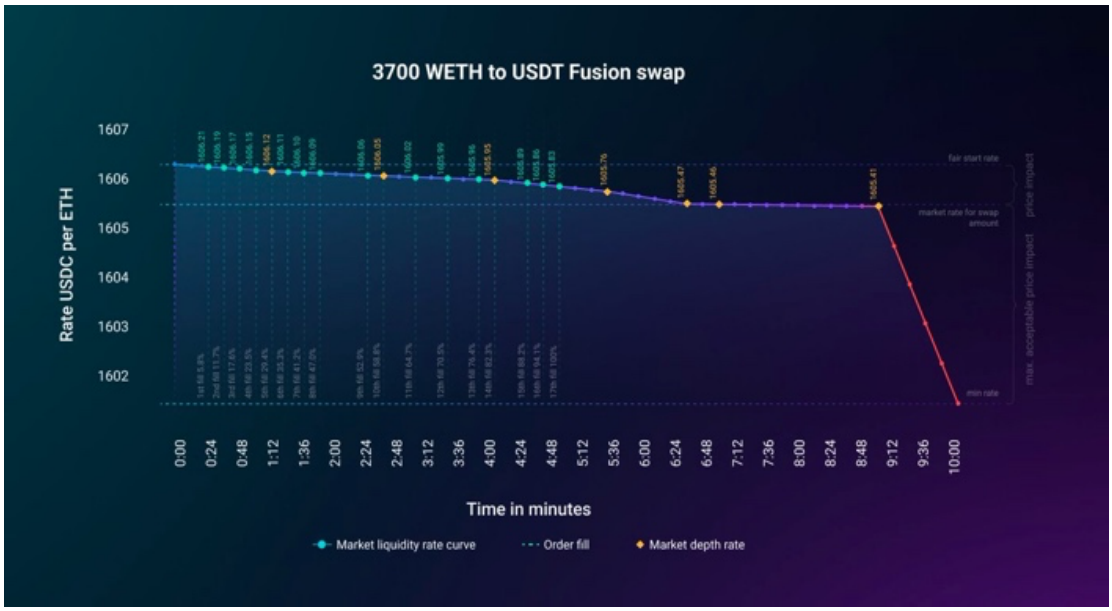

Figure 2. 1inch Fusion Price Curve

Due to the handling of the execution by the solvers directly, they can factor in slippage and gas costs into their solutions effectively. This also creates a clear understanding for a Fusion solver what margin they can extract. Such rewards are received by the solver immediately on a per transaction basis.

In contrast with CowSwap, Fusion results in competition occurring on-chain as solvers attempt to outbid gas costs in order to be picked over competitors by builders and relayers. This ultimately results in a sacrifice of user price and solver reward, with value being redistributed to network validators.

2.1 MDEX Design Challenges

By exploring existing solutions we have identified the following challenges in designing MDEX systems:

- The determination of an accurate price to the user
- Ensuring solver competition maximises value add to users and solvers rather than network validators
- Creating a clear and predictable incentive structure for solvers.

3. JAM (Just-in-time Aggregation Model)

3.1 Bebop JAM Overview

Bebop JAM is a MDEX that uses an RFQ based system to determine prices and respective solutions. It also delegates the execution of each order exclusively to the winning solver. This combines the benefits of competition occurring off-chain and execution being handled by solvers.

From a systems perspective, how the trade is conducted can be defined as follows *(Fig 3)*:

1. Bebop JAM orchestrator receives an order request in the form: `Buy 1000 Token A - Sell Token B - Slippage: 0`. The requests support orders for multiple tokens being exchanged simultaneously.
2. It queries solvers for a solution to the trade; the best solution is returned to the user.
3. The user signs and therefore confirms their trade, producing a signature over the winning solver's solution.
4. This signature is returned to the orchestrator.
5. The orchestrator passes the user's signature to the winning solver for them to execute on-chain.
6. The solver returns the transaction to the orchestrator and eventually the user.

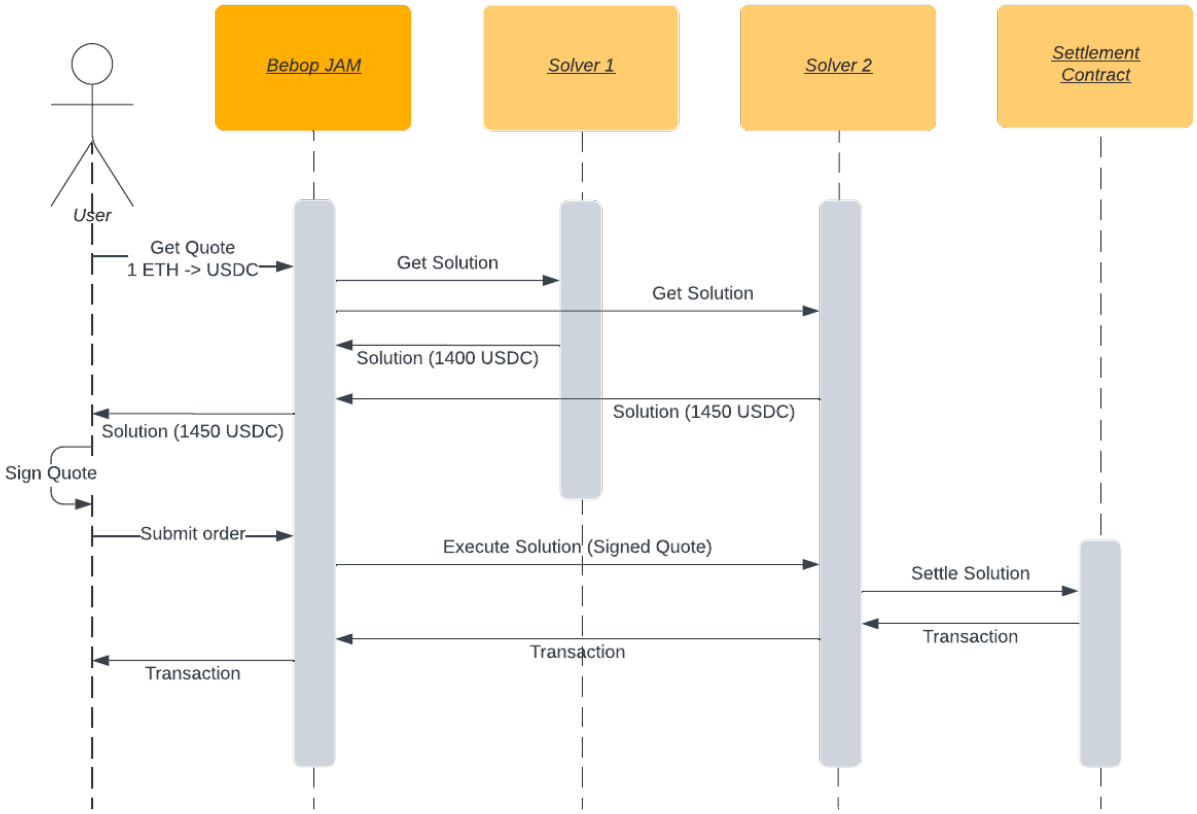

Figure 3. Sequence of Interactions in Bebop JAM

Several improvements become apparent as a result of this model.

Firstly, solvers can be queried for an exact quote that they can reliably execute, where users receive a fixed price and certainty. This is in contrast to the models where the limit orders posted may sit idle with the orchestrator during adverse price movements on-chain, or in the case of 1inch's Fusion, high gas costs potentially creeping into the price at execution.

Additionally, instead of relying on payments from the protocol in the form of gas costs and token rewards, solvers have clear economic incentives defined on a per transaction basis. This creates a straightforward framework for solvers in terms of margins and thus ensures that they can offer the most accurate prices to users.

Most importantly, as the competition between solvers happens solely off-chain, sacrifices of the theoretical best price to validators for priority is avoided. Consequently, this model results in a surplus. available to improve price for the user and rewards for the solver.

3.2 Solver API

A rough outline of a solver API is thus presented. The Bebop JAM system can communicate with solvers over http or websockets.

3.3 Get Solution

Given a quote request, return the best solution.

Example Request:

```
{
   "id": "123",
   "sellTokens": ["USDC", "DAI"],
   "buyTokens": ["WETH"],
   "sellAmounts":["10000", "1000"],
   "slippage": 0,
   "receiver": ["0x34..."]
}
```

The solver must return a solution in the form of `buyAmounts` or an empty response to indicate that a solution cannot be made.

Example Response:

```
{
   "id": "123",
   "buyAmounts": ["6.9873789"],
   "interactions":
   [
     {
       "to": "0x9522229…",
       "data": "0x000...",
       "value": "0x0"
     },
     {
       "to": "0x3a30af…",
       "data": "0x000...",
       "value": "0x0"
       }
   ]
}
```

3.4 Execute Solution

Given a quote and user's signature - execute the solution immediately, returning the tx hash.

Example Request:

```
{
   "id": "123",
   "signature": "0x123456...",
   "quote": ..., // As per "get solution"
   request
   "solution":  ...  //  As  per  "get
   solution" response
}
```

Response: `{ "tx_hash": "0x123456..." }`

The solver executes the transaction on-chain against the JAM Settlement contract:

```
function settle(JamOrder.Data calldata
order, JamInteraction.Data[] calldata
interactions)
```

3.5 Incentivising Solver Reliability

One side effect that can occur as a result of moving competition off-chain but maintaining independence in execution by solvers is that as a result of constantly changing on-chain liquidity, and other motives, they may ignore or fail execution. Ultimately the solvers will want to continue receiving order flow and as the solver set will initially be permissioned, they have every incentive to act honestly. Where the solver set becomes permissionless over time, there are several methods that can be employed to incentivise reliability.

Firstly, the solver is incentivised to produce solutions that succeed on-chain. Given that execution costs are incurred by the solver themselves.

In addition, to prevent solvers from holding off on execution or failing transactions at low costs (such as on-chains with low fees) a scoring system can be used. The execution score adjusts with their reliability on factors such as response speed, execution speed and success rate. Solvers with low scores will be deprioritised in the best solution function. Reliability scores have been successfully employed with permissionless infrastructure, such as Flashbots searcher scores [3].

REFERENCES

[1] DefiLlama

[2] Coincidence of Wants

[3] Searcher Reputation | Flashbots Docs