Feedlot: trading with safe, low latency price oracles [*]

The Feedlot working group

> *feed* — Something supplied continuously.
> *lot* — One or more items auctioned or sold as a unit, separate from other items.

Purpose of this document This article is a feasibility study for a class of automated market makers (AMMs) that passively provide liquidity at a price based on the uniform clearing price (UCP) of a competitive batch auction.

It is not a whitepaper, specification, or proposal, and it makes no claim to exhaustiveness or completeness on any of the topics discussed — particularly security. Would-be implementers of a feedlot AMM are advised to pursue thorough further investigations of these issues in the context into which they hope to deploy.

[*]This research was funded by the Gnosis DAO.

1 Introduction

Feedlot AMMs should satisfy the following constraints:

- Feedlot AMM LPs should enjoy cheap portfolio management, some yield, and protection from adverse selection.
- Feedlot AMM traders should enjoy low, predictable fees, control over execution time, and favourable prices at least for trades in the 'correct' direction.
- It should not be economical to manipulate the UCP of the CoW batch auction in order to trade at favourable prices on feedlot.

  Moreover, it should not be possible to substantially offset the cost of manipulating the CoW UCP (for any reason) by trading on a feedlot AMM.
- Wherever possible, Feedlot should use incentive-compatible mechanisms to ensure correct operation, without needing to fall back on social adjudication procedures.
- Feedlot provides a new revenue stream for CoW protocol via oracle fees.

2 General design principles

2.1 Definitions

Our blockchain model is based on Ethereum, and should apply to any blockchain-based state machine with similar principles. In particular we point out the following assumptions:

- Accounts model that tracks token balances with a common interface (e.g. ERC20) and such that token transfers satisfy usual invariants (i.e. transfers must preserve total supply and balances cannot be negative).
- Transactions (a.k.a. messages calls) must be initiated by an off-chain entity.
- Transactions, once committed, cannot be rolled back. That is, we do not consider the risk of blockchain forks.

By a *smart contract system* we mean any kind of on-chain entity or aggregation of entities. In particular, it may mean a single smart contract (e.g. a Uniswap pool) or a structured collection of interacting smart contracts (e.g. Uniswap as a whole).

- *Liquidity pool.* Smart contract system that custodies the assets of *liquidity providers* and tracks withdrawal liabilities.
- *Liquidity provider (LP).* An agent that deposits funds in a liquidity pool.
- *Automated market maker (AMM).* Smart contract system that provides pricing and passively settles orders by trading against a liquidity pool.
- *Batch auction.* Order settlement system that accumulates orders in a buffer and settles them with a uniform clearing price. Prices are supplied by *solvers* who compete to optimise an objective function defined in terms of the price vector and the set of orders.
- *Uniform clearing price (UCP).* Price vector against which a batch auction is settled.
- *Solver.* Agent that provides quotes in competition to settle a batch auction.
- *Constant function market maker.* AMM whose pricing function depends only on reserves.
- *No-arbitrage price.* A price p such that it is not possible to to make a

In this article, the pricing provided by an AMM will be a function of state at the time of settlement. In principle it is possible for it to also depend on calldata in the message that triggers the settlement. For simplicity, we will mainly consider only pools with two assets A and B whose balances are denoted $(x,y) \in [0,\infty)^2$. We take token A for the numéraire (also called the 'quote token'), so that prices are for token B in terms of token A. If a trade occurs of Δx A tokens for Δy B tokens, the execution price is $\Delta x/\Delta y$.

2.2 Constant function market makers

By far the most widely used type of AMM is a constant function market maker (CFMM). A CFMM is defined by its *invariant function* $f(x,y)$, a real-valued function of the pool reserves. The associated marginal pricing function is

$$p(x,y) = \frac{f_x(x,y)}{f_y(x,y)}.$$

If reserves are in state (x_0, y_0), the amount of B tokens paid out in exchange for $-\Delta x$ A tokens is

$$\Delta y = \int_{x_0}^{\Delta x} p(x, y(x))dx - y_0$$

where $y(x)$ is determined by the corresponding differential equation $y'(x) = p(x, y(x))$ and the initial conditions $y(x_0) = y_0$.

A practical implementation of market orders on a CFMM must have a computationally efficient algorithm for computing Δy given Δx. On the other hand, to decide if a given swap $(\Delta x, \Delta y)$ will be accepted by a CFMM, it is enough to compute the invariant $f(x+\Delta x, y+\Delta y)$.

2.3 Liquidity provider costs

The CFMM quote is not automatically updated when changes occur on other markets. Rather, price updates are supplied by arbitrageurs who trade on the CFMM in such a way as to push the pricing into alignment with external markets.

This trading activity also has the effect of adjusting the balance of assets in the pool, computed in terms of external market prices, towards an equlibrium that depends on the invariant. For example, with Uniswap's constant product formula $f(x,y) = xy$, the equilibrium balance is 50-50, that is, $px = y$.

LPs effectively pay for this adjustment service by trading at unfavourable prices. The amount they pay depends only on the external price movement. In particular, there is no room for price updaters to compete with one another on the basis of reducing cost to LPs; rather, all arbitrageur competition is at the infrastructure layer and excess rewards tend to be absorbed either by consensus nodes or by middleware services that assemble blocks or partial blocks. cite

2.4 Automated market makers with a price feed

The issue of uncontrolled LP losses to adverse selection could be alleviated if price updates from external markets could be supplied by another channel, so that the marginal pricing function could depend on inputs other than reserves:

$$p = p(x, y; p_{\text{ext}}).$$

In blockchain applications, such channels are called *price oracles.* cite

Example. The most obvious choice for an oracle-based pricing function is

$$p(x, y; p_{\text{ext}}) = p_{\text{ext}},$$

so that the AMM simply quotes the oracle price. We refer to this as *passthrough* pricing. If we black-box the data source and assume that it provides no-arbitrage prices, then such an AMM would indeed be protected from adverse selection.

However, as is well-known, oracles in the field are fallible. They are vulnerable to bugs and deliberate attacks at each stage of the data pipeline. Oracles that provide prices that anyone can trade against are a particularly attractive vector for sabotage, both because of the potential for financial gain and because, in the case of a manipulated data source, of the difficulty in establishing that an attack has even occurred. cite

Moreover, simply passing through the oracle price offers no control over inventory management. In particular, if trading tends to occur more on one side than the other for whatever reason, reserves of the in-demand asset could easily dry out **garman1976market**, §2.2. In traditional finance, this risk is typically accounted for by adjusting pricing to be more or less favourable depending on whether a trade would push reserves towards or away from the desired inventory **biais2005market**. Assuming a typical microeconomic environment with negatively sloped demand curves, order flow naturally skews towards the buy (resp. sell) side as prices edge below (resp. above) the market clearing price. Equally, one can suppose that a population of arbitrageurs able to trade frictionlessly at the oracle price will effect the desired inventory adjustments.

Example. Suppose we target a reserve balance of 50-50 (computed in terms of the oracle price), setting

$$p(x, y; p_{\text{ext}}) = p_{\text{ext}} \cdot g\left(p_{\text{ext}} x/y\right)$$

where $g : (0,\infty) \to (0,\infty)$ is a monotone increasing function such that $g(1) = 1$. The most basic choice is $g(r) = r$, in which case the price oracle drops out of the formula and we recover the UniswapV2 pricing function $p(x, y; p_{\text{ext}}) = x/y$.

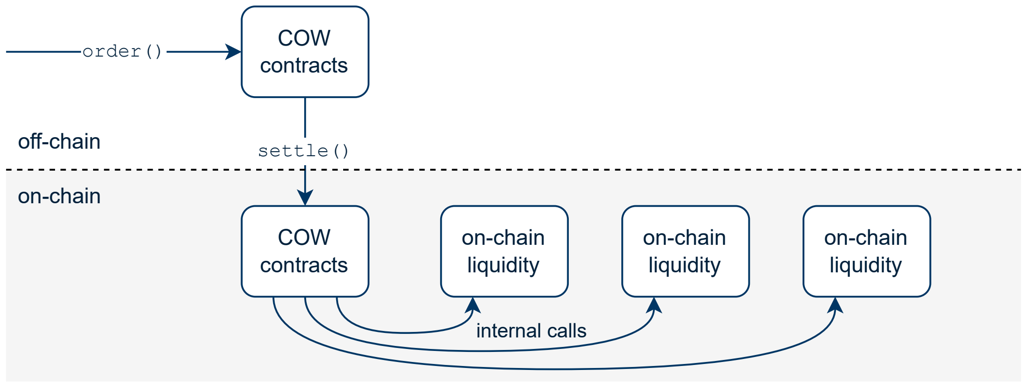

Figure 1: CoW procotol execution. CoW services accumulate orders and batches are settled (at most) every 30 seconds.

Example. More interesting is to choose a function which has a small or vanishing gradient near 1. A simple example is

$$g(r) = \begin{cases} re^{\tau} & r \leq e^{-\tau} \\ 1 & -\tau \leq t \leq \tau \\ re^{-\tau} & e^{\tau} \leq r \end{cases}$$

where $\tau > 0$ is the 'tolerance' of the pool for reserves to deviate from an even balance. It is straightforward to compute integrals of these quantities, hence execution prices, analytically.

Example. Say something about the stableswap curve. todo

3 Architecture

Our model for a batch auction system is the CoW protocol on Ethereum. The CoW protocol consists of a set of off-chain entities that we collectively refer to as *CoW services* and a system of Ethereum smart contracts called the *CoW contracts*. The details of the internal architecture of these aggregations is not discussed here.

Schematically, the CoW algorithm runs as follows:

1. Traders send orders to an off-chain order book server where they are tracked in a database.
2. A set of offchain entities called *solvers* query the database and attempt to construct a *solution*, that is, an Ethereum transaction that collectively settles all user orders at a fixed price vector $\vec{p}$ (the uniform clearing price). This transaction may contain arbitrary Ethereum message calls.
3. Every batch interval, solutions are validated by simulating against a recently observed chain state and ranked according to a utility function. The utility function measures the total marginal utility of user orders filled by the solution in terms of the difference between the limit price and the fill price. Utilities of orders denominated in different tokens are aggregated using prices on external markets.
4. The solver with the winning solution calls the `settle()` function on the CoW settlement contract, which executes their solution.
5. Misbehaviour is assessed socially. Slashing punishments can be triggered by a vote of the CoW DAO.

An AMM whose quote depends on the UCP of the CoW batch auction needs to have at least the following structures:

- A way for traders to submit orders.
- A way for would-be LPs to add or remove liquidity.
- A communication channel $\mathcal{O}$ connecting Feedlot with the CoW contracts or services along which can be communicated the UCP.

Since the last item is the distinguishing feature of Feedlot AMMs, in this section we focus on the design of this channel.

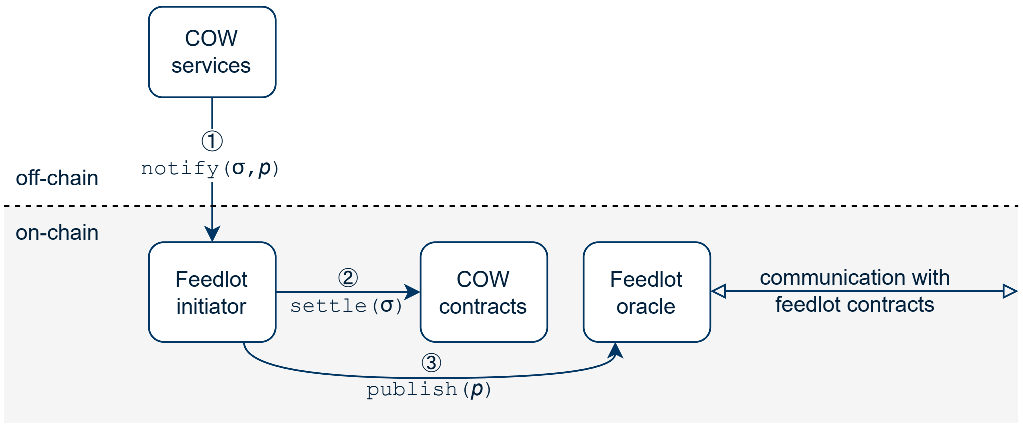

Figure 2: Atomic execution of the batch solution and publication of the UCP. The ordering of (2) and (3) can be reversed.

3.1 Synchronisation

A batch auction system needn't be synchronised with the blockchain. There may not be an auction every block, and even if there is, not every token pair traded on the BAS need necessarily appear in the batch. This is the case for CoW: auctions are roughly every 30 seconds, and sometimes no auction occurs for hours at a time. Therefore, a fresh batch auction price might not be available at the point that a trader wishes to trade on feedlot.

If the Feedlot quote price is to depend on the price of a CoW batch auction, then, there are only two possibilities:

1. The quote function uses a stale CoW price (poll model).
2. The market only settles trades at the same time as the batch auction (subscription model).

These two options are analogous to what in the blockchain oracles literature have been called the *pull* and *push* models, respectively **heiss2019oracles**; **muhlberger2020foundational**.

In either case, CoW services can guarantee zero latency updates of the UCP by writing into the channel atomically together with the batch execution transaction. This would require minor modifications to CoW's offchain components so that they call a wrapper contract that both triggers the CoW settlement and publishes a price to the on-chain oracle. It does not require any changes to CoW's onchain components. Whether the UCP is updated before or after the settlement affects only how Feedlot behaves for orders placed in the winning solution itself, with the details depending on the synchrony model for Feedlot; see §3.2. Orders that arrive at Feedlot outside the batch are not affected by this ordering.

What could be considered a third option is to fall back on another algorithm in case no price is available. Since that essentially amounts to not trading on Feedlot, and could easily be implemented at a higher layer, we don't discuss that here.

3.1.1 Poll model

In this model, CoW writes the UCP vector to the channel every time there is a batch. Meanwhile, Feedlot AMMs query the price whenever they have to settle an order. There is no synchronisation between these two tasks. One write is needed per batch per product, and one read is needed per trade on Feedlot. These operations being relatively cheap, it would be reasonable to implement this type of channel as an on-chain oracle. A variant approach is to aggregate UCPs to produce, for example, time-weighted average prices. This is the approach taken by Uniswap's oracle service **adams2020uniswap**.

3.1.2 Subscription model

In this approach, orders arriving at Feedlot are accumulated in an order book. All trades are executed when there is a CoW batch.

As with any order book based protocol, the order book itself would potentially be quite expensive to maintain on the Ethereum chain. It is beyond the scope of this report to discuss approaches for optimising trust model and cost-effectiveness of off-chain order books. Pragmatically, a reasonable approach would be to maintain the order book in the same place as the order book for the CoW batch auction, which at time of writing is a set of servers maintained by the CoW team.

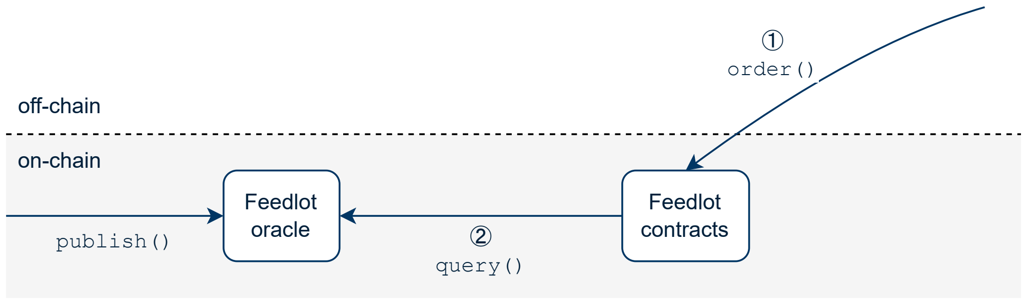

Figure 3: Polling oracle. Feedlot AMM executions are not synchronised with CoW batches.

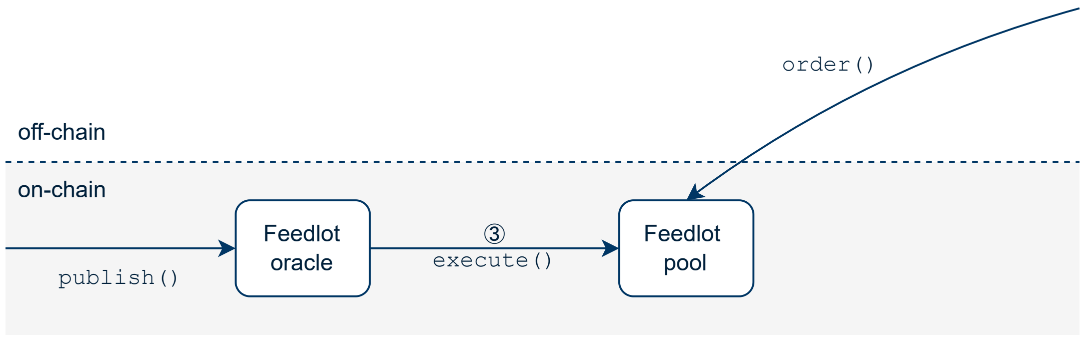

Figure 4: Subscription oracle. Feedlot AMM executions are triggered when a new price is published.

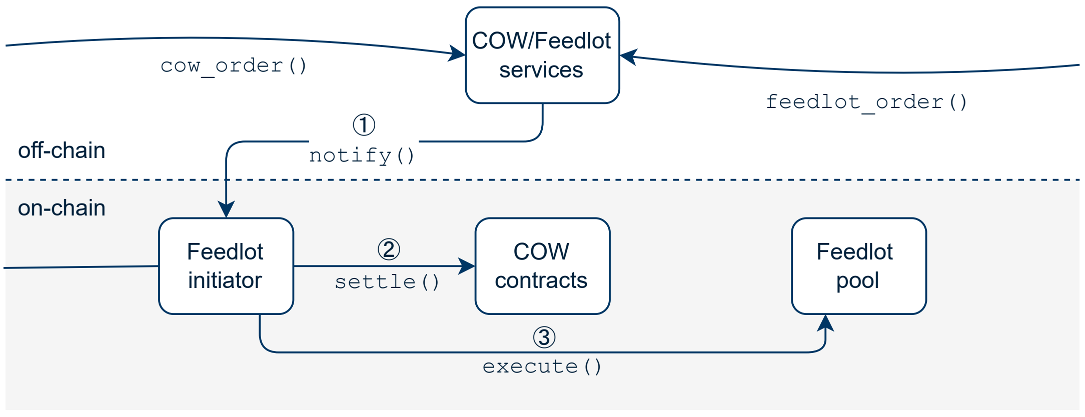

Figure 5: Integrated oracle with offchain order queue. Feedlot AMM executions are triggered when a new price is published.

3.2 Can the batch itself settle on Feedlot?

Certain configurations would allow the solution to both settle orders on Feedlot and quote it a price. The solver's objective is to get the most favourable price possible for its users, so naturally it will quote the least favourable price that Feedlot will accept. Naturally, allowing an entity to trade against the AMM at any price it chooses is trivially exploitable to the detriment of LPs, and so special care has to be taken to address this case.

3.2.1 Poll model

In the poll-based model,

- If the UCP is updated *after* the batch, the solution trades with a stale price.
- If the UCP is updated *before* the batch, the solution trades at the price quoted by the solver.

The interesting case is the second one. Assuming solutions are simulated exactly as they would run on-chain, a solver could quote any price at all and have it execute successfully against Feedlot. The solver's objective is to get the most favourable price possible for its users, so naturally it will quote the least favourable price that Feedlot will accept. Naturally, allowing an entity to trade against the AMM at any price it chooses is trivially exploitable to the detriment of LPs. Thus, if this approach were to be pursued, a special mechanism would have to implemented where Feedlot is specifically excluded from simulation.

Let us discuss, and dismiss, in turn a few approaches to making this work:

- Bound Feedlot's pricing function below independently of the UCP. For example, accept a trade only if it offers better pricing for the pool than some hardcoded aggregate of prices from whitelisted external markets. Then rational solvers will always trade at exactly this lower bound. This eliminates the dependence of pricing on the UCP.
- Adjust the solution objective to some strictly convex function that takes into account utility of the pool as well as active traders.[1] The objective function will have to be normalised so that, independently of absolute price, it achieves some desired balance between pool and trader utility. Clearly, this price normalisation cannot be derived from the price quoted by solvers, because as before they would then be free to set it arbitrarily. Hence, an auxiliary price oracle is needed.
- Allow solutions that successfully simulate on markets other than Feedlot to actually settle instead on Feedlot, at the price that was realisable on those markets. This has the effect of making price manipulation costless: solvers can provide their own liquidity at any price, knowing that they will not be held to trading at that price, this burden ultimately falling on Feedlot LPs.

It is possible that some combination of these and other approaches could allow solvers to safely both trade and quote on the Feedlot liquidity pool.

3.2.2 Subscription model

Batch solutions must settle immediately, so the success of the solution cannot depend on trading on an AMM that enqueues orders for execution at a later time. However, a batch can be used to atomically execute any message call. A solver could therefore use the following strategy:

1. Fill orders using private liquidity.
2. Enqueue orders in the opposite direction on Feedlot.

Financially, this amounts to hedging the trade in step 1. with a forward contract with expiry at the next batch. In this way, Feedlot could act as a kind of repo market for solvers.

4 Cryptoeconomic security

A price oracle is a data source $\mathcal{O}$ from which a smart contract may query prices for a market order

$$p \leftarrow \mathcal{O}(t_{\mathrm{in}}, t_{\mathrm{out}}, \sigma, r)$$

where $t_{\mathrm{in/out}}$ denotes the in/out token, $\sigma \in \{\mathrm{BUY}, \mathrm{SELL}\}$ is the direction, and r is the amount (of t_{in} for SELL orders, t_{out} for BUYs). The oracle is said to be *truthful* if the prices thereby obtained are free from arbitrage.

Assuming correct implementation, oracles can fail in two ways:

- Inaccurate or unreliable data source.
- Failure to faithfully communicate the value from the data source.

In the present context, fidelity of the communication of value is guaranteed by non-falsifiability of on-chain execution.

[1] If it isn't *strictly* convex, solvers will be indifferent between delivering utility to traders or the pool.

4.1 Truthfulness of pricing data

We say that the price p is *realisable* for an agent I at a volume V of the in token and chain state ϕ if it is possible for I to buy V A tokens for pV B tokens in state ϕ. This is a way to make sense of the 'truthfulness' property for oracles sketched in **heiss 2019oracles**.

If CoW solver prices are not realisable, the solver must settle the trade with their own private liquidity held in bond by CoW DAO. A posteriori, CoW solver prices are realisable for the solver at the state immediately before the batch and at the volume traded in the batch. That is, the structure of the CoW batch auction itself guarantees that the UCP vector is realisable at the volume traded in the batch.

It is worth noting that merely simulating executions on-chain (and reverting all changes at the end) is sufficient to verify realisability of the price vector.

4.2 Manipulation resistance

Wherever economic decisions are made based on the output of an oracle $\mathcal{O}$, there lies an incentive to effect changes in the value published by $\mathcal{O}$ in order to influence those decisions. In few settings is this incentive more transparent than the present case of a price oracle coupled with an automated market that offers to trade at or near that price. If, for example, some party is able to manipulate $\mathcal{O}$ at a cost C_{manip} so that it publishes a price p that is strictly less than a price p_{ext} realisable on some other market, then a strategy of doing so and then buying a quantity V of the risky asset on Feedlot yields a profit of

$$V \cdot (p_{\text{ext}} - p) - C_{\text{manip}}.$$

Even if this quantity is negative, trading on Feedlot in this way *offsets* the cost of manipulation, which may still be strategically optimal because of other decisions contingent on the output of $\mathcal{O}$. Note that this type of activity does not undermine the fidelity of the oracle itself: a manipulated price may still be realisable.

In general, it is difficult to distinguish price manipulation from other types of economic activity that affects prices **ky le2008define**; **zhang2022competition**. However, it is quite straightforward to define **manipulation resistance**: a game is manipulation resistant if the cost of effecting a change in the oracle price away from the prevailing market price is greater than the marginal profit that can be made by exploiting such a price change:

$$C_{\text{manip}}(V, p, p_{\text{ext}}) \gg U(\sigma; p) - U(\sigma'; p_{\text{ext}}).$$

Here σ is a strategy that is optimal at oracle price p and σ' is an optimal strategy at oracle price p_{ext}.

To analyse this property concretely, it is necessary to have models for the strategy space.

- To cause the CoW UCP to be equal to $p < p_{\text{ext}}$ at volume V, it is necessary and sufficient to cause a volume V of the risky asset to be available for purchase by a solver at price p. Ignoring second-order effects such as the reaction of other market participants to a price change, the basic cost of this is $V' \cdot |p_{\text{ext}} - p|$, where $V' \leq V$ is the volume of user orders in the batch not placed by the manipulator.

- To model the marginal proceeds of manipulation, we consider only the direct profits from trading on Feedlot at favourable prices, that is,

  $$V_{\text{Feedlot}} \cdot \left|p - p_{\text{ext}}\right|$$

  where V_{Feedlot} is the amount traded. Other profits from as-yet unrealised uses of $\mathcal{O}$ are difficult to predict. CoW or would-be Feedlot developers may have some ability to constrain, or at least channel, decision-making based on $\mathcal{O}$ by restricting on-chain access. However, the feed would still be visible to anyone with access to an Ethereum node.

With these models, the essential condition for manipulation resistance is

$$V_{\text{Feedlot}} < V'_{\text{CoW}}.$$

If V'_{CoW} can be estimated, this inequality can be enforced in the form of a volume limit for Feedlot. Unfortunately, it is hard to calculate V' in practice, because orders owned by an agent following a manipulation strategy cannot easily be distinguished from the orders of other agents. The potential for low-cost wash trading means that in general, it need not be possible to estimate V' in terms of V.

5 Economic impact

5.1 For LPs

In §2.4, we characterised the introduction of a price feed to our AMM pricing function as a way to mitigate or prevent LP losses due to adverse selection. Can we quantify the effects of this mitigation in practice?

In **milionis2022automated**, the authors introduce a framework to quantify losses of CFMM LPs to adverse selection that depends on an external market price P. Adapting this framework to discrete time and with the CoW UCP playing the rôle of the external market price, we can define the *CoW-based loss-versus-rebalancing* as

$$\mathrm{LVR}_n := R(P_n) - V(P_n)$$

where:

- P_n is the UCP of the nth CoW batch;
- $V(P_n)$ is the optimal CFMM pool value at an external market price of P_n;
- $R(P_n) = V_0 + \sum_{i=1}^{n} y^*(P_n)\cdot(P_n - P_{n-1})$ is the value of the self-financing rebalancing portfolio at the time of the nth CoW batch, where rebalances take place only at CoW batch times. (If trading fees are negligible, path-independence means that it doesn't actually matter when rebalancing takes place.)

Discrete-time analogues of the arguments of *op. cit.* show that this is a non-negative and non-decreasing process.[2] In continuous time, the quantity can be expressed in terms of the instantaneous square volatility $\sigma^2(P)$ and $V''(P) > 0$; one can derive similar, though uglier, formulas for the discrete case.

If Feedlot uses passthrough pricing and accepts trades in such a way that its reserves track the rebalancing portfolio, then R_n can also be interpreted as the portfolio value of a Feedlot LP. That is, LVR_n is the difference in performance between a Feedlot LP and a CFMM LP. This is contingent on a sufficient 'uninformed' order flow arriving at Feedlot for it to track the reference portfolio by selectively accepting trades.

[DATA STUDIES GO HERE]

5.2 For traders

Feedlot LPs take up a short position on a kind of option to trade at a certain oracle price. But by construction, this price is also realisable on other markets; that is, this optionality already exists 'naturally.' Why then should traders be interested in purchasing this product?

- *Protection.* By trading along with the CoW batch, the risk of price changes between commitment and execution time (whether due to 'natural' variability or deliberate attack) is socialised across the whole CoW/Feedlot execution. Traders on CoW itself also enjoy this benefit.
- *Spread.* User orders on the CoW batch auction employ solvers to actively seek out favourable prices. On the other hand, market orders on Feedlot simply wait for someone on CoW to ask for a price for that pair and then take advantage of the result. Feedlot LPs do no price-finding work, and depending on how the liquidity curve is configured, may also enjoy portfolio management services (at the cost of unpredictable execution prices for users). Hence, Feedlot trading fees (or equivalently, the bid-ask spread) must be cheaper than the active service provided by CoW protocol.

6 Conclusion

A team wishing to implement a Feedlot AMM should take great care to ensure that it does not make the CoW UCP an economically viable target for manipulation. In §**??**, we have established that some kind of volume limit is a necessary condition for this. However, naïvely limiting volume in terms of the volume on CoW is not sufficient, since this quantity is itself prone to relatively low-cost manipulation.

Social enforcement The solver itself is the most privileged actor in this game. While this places it in a uniquely powerful position to manipulate CoW prices itself — and therefore ought to be the subject of particular scrutiny — it also is uniquely positioned to fight *against* manipulation by arbitraging manipulated prices into line with external markets in its solution. However, fighting against manipulation is not necessarily aligned with the objective function currently in effect on CoW protocol. As it stands, CoW solvers are already economically exposed to the governance decisions of the CoW DAO.

Incentive-aligned manipulation resistance In some cases, trading against a would-be manipulator may be economically viable even without social enforcement. Suppose a wash trader submits a large volume of liquidity orders in both directions. Suspecting a manipulation attempt, the solver discards the orders in one direction and himself takes the opportunity to make a large directional trade with no slippage. Depending on the market depth, this might be very

[2] We lose the predictability property, because of course, the discrete-time analogue of a diffusion process is not predictable.

References

adams2020uniswap Hayden Adams, Noah Zinsmeister, and Dan Robinson. *Uniswap v2 core*. 2020. URL: https://uniswap.org/whitepaper.pdf.

biais2005market Bruno Biais, Larry Glosten, and Chester Spatt. “Market microstructure: A survey of microfoundations, empirical results, and policy implications”. In: *Journal of Financial Markets* 8.2 (2005), pp. 217–264.

garman1976market Mark B Garman. “Market microstructure”. In: *Journal of financial Economics* 3.3 (1976), pp. 257–275.

heiss2019oracles Jonathan Heiss, Jacob Eberhardt, and Stefan Tai. “From Oracles to Trustworthy Data On-Chaining Systems”. In: Atlanta, GA, USA. Atlanta, GA, USA: IEEE, 2019, pp. 496–503. ISBN: 978-1-7281-4694-2. DOI: 10.1109/Blockchain.2019.00075.

kyle2008define Albert S Kyle and Sathish Viswanathan. “How to define illegal price manipulation”. In: *American Economic Review* 98.2 (2008), pp. 274–279.

milionis2022automated Jason Milionis et al. *Automated market making and loss-versus-rebalancing*. 2022. arXiv: 2208.06046 [q-fin.MF].

muhlberger2020foundational Roman Mühlberger et al. “Foundational oracle patterns: Connecting blockchain to the off-chain world”. In: *Business Process Management: Blockchain and Robotic Process Automation Forum: BPM 2020 Blockchain and RPA Forum, Seville, Spain, September 13–18, 2020, Proceedings 18*. Springer. 2020, pp. 35–51.

zhang2022competition Anthony Lee Zhang. “Competition and manipulation in derivative contract markets”. In: *Journal of Financial Economics* 144.2 (2022), pp. 396–413.