In this post, we'd like to outline a technical proposal of how we can permissionlessly support flash-borrowing of funds in order to enrich the power of CoW Protocol settlements.

Flash loans could be useful for a variety of integrations, that want to "bridge" or "wrap" an existing "searcher based" trading model to make it CoW Protocol compatible without releasing a new version of their smart contract.

These include for instance:

- Liquidation engines such as Aave & Compound
- Levering up/down collateralized debt positions (e.g. Oasis)
- AMMs that want to implement surplus capturing trading behavior

The main benefit of using the CoW Protocol model in these cases is that it allows the users or the underlying protocol to capture any price improvement (aka surplus) from getting settled at the "competitive market price", rather than paying price incentives to MEV bots who ultimately bribe Ethereum validators for the right to access a favorable trade.

CoW Protocol Settlement Process

[

image

1218×424 63 KB

](https://europe1.discourse-cdn.com/business20/uploads/cow/original/1X/989555a8178e4a2e0e0d6d7eaf24407a6b11f87a.jpeg)

The way that CoW Swap settlements work at the moment is that:

1. The winning solver invokes the settle method on the CoW Settlement contract
2. Each order can invoke a set of so-called "pre-interactions" to prepare for the trade (e.g. set an allowance)
3. All sellTokens from all CoW Swap orders are transferred into the settlement contract
4. The solver invokes a list of "interactions" converting the funds in the settlement contract to create the required amount of output tokens to fulfill the settlement
5. The buyTokens are paid out to the recipients (ensuring that the limit price is met)
6. Each order finally has the opportunity to invoke arbitrary "post-interactions" (e.g. move funds into a bridge contract)

Note, that all these steps happen in the same atomic transaction.

Specifically, this means that the order in which interactions are executed relies on the fact that some initial capital - all sell amounts - is available (in fact it is a non trivial side problem of being a solver to correctly order interactions in a complex CoW settlement without temporarily going negative in some token, which would cause a revert)

The need for Flash Loans

While e.g. liquidations could trivially be implemented using this model (place a sell order from the CDP if some minimal collateral ratio is reached and know that - if the leave the contract - in the same transaction you will receive at least what you asked for), we see that in a lot of use cases today protocols allow specific addresses to unlock the collateral only if they pay back the loan in the same atomic call context (not just the same transaction).

E.g. the logic for rebuying collateral in compound can be seen here:comet/Comet.sol at 729a7a26c63a39a63416775f350a9554071e02e0 · compound-finance/comet · GitHub

[

Screenshot 2023-05-25 at 09.43.12

1250×820 186 KB

](https://europe1.discourse-cdn.com/business20/uploads/cow/original/1X/88f00e14bf332b96f204428a1c3574fe4cf8c368.jpeg)

Since CoW Protocol requires to move funds from the user first, then exercises a bunch of calls before it pays out the proceeds, this model isn't easy to integrate. However, if we could flash borrow the loan currency first (e.g. in a pre-

interaction), we could use it to unlock the collateral into an intermediate CoW Protocol smart contract order (using another pre-interaction), then use it in the settlement to recoup the loan currency (with surplus!!!)

and finally pay back the loan in a post interaction.

In fact, we have seen Flash Swaps implemented by the Agave team to allow for their liquidations to be compatible with CoW Protocol (e.g. Gnosis Transaction Hash (Txhash) Details | GnosisScan). However, this design involved a custom token which grants flash loans across call-stacks and assumes off-chain guarantees on pre- and post-interactions being called “in lock-step” (which the current smart contract doesn’t guarantee).

[

image

1252×416 21.1 KB

](https://europe1.discourse-cdn.com/business20/uploads/cow/original/1X/3335c351c8ccbe3f797c80f8d5c615b1355fa91c.png)

The main limitation here is the current requirement for a solver to use the settle

method (step 1 above) as an entry point.

If instead solvers were able to take out a loan before they invoke settle

(and instead have settle

being invoked in the flash-loan callback), they could fund the CoW order using the loan and at the same time guarantee that the settlement would fail if the loan wasn’t ultimately paid back in the post-interaction step.

[

image

1292×442 16.9 KB

](https://europe1.discourse-cdn.com/business20/uploads/cow/original/1X/b91ffc073e6a7bd7a451d5331baecd889ab1905f.png)

This could be achieved by a wrapper contract, which first invokes a flashloan method and then provides a callback to initiate the CoW Settlement. This way the CoW Settlement gets executed within the call-stack of the flash loan and has access to atomic guarantees such as reverting if the loan is not paid back in the end. Note, that the same pattern could also be used to enforce other behavior that should happen in lockstep (e.g. lowering and resetting a LP fee parameter on an AMM)

The wrapper contract itself would become an allow-listed solver contract in the protocol (required for it to call settle) and itself enforce the same solver authentication as the main settlement contract (ie only bonded solvers are allowed to invoke it)

Below you can find some very rough unaudited sample code of how this wrapper contract could look like (h/t @fedgiac & @nlordell):

contract Wrapper { GPv2Interaction.Data public callback;

function callWhileExpectingCallback(GPv2Interaction.Data call, GPv2Interaction.Data _callback) external onlySolver { // callback is a settlement for us, call is what we want to do that triggers // any kind of callback to this contract (leading to fallback handler being invoked) callback = _callback; GPv2Interaction.execute(call); require(!callback) }

fallback() external nonReentrant { require(callback); GPv2Interaction.execute(callback); delete callback; } }

Please feel free to comment with ideas, concerns or thoughts on this proposal. We are also always happy to work with community members on implementing this in the context of a grant.