Abstract

This topic introduces a method for purchasing gas on Ethereum. Technically it is a fully on-chain Ethereum gas futures market. The volatility of gas prices, primarily driven by network demand fluctuations, significantly obstructs user experience on Ethereum due to unpredictable costs. This solution aims to mitigate that issue.

Protocol Description

General

The protocol users might be divided into to parties, gas purchasers (Purchaser) and gas providers (Executors)

The general protocol workflow has these stages:

1. Purchaser: Listing order
2. Executor: Accepting order
3. Purchaser: Requesting transaction execution
4. Executor: Executing transaction
5. Anyone: Liquidating executor

Listing order

It is required to place order conditions onchain regarding the execution timeframe, gas price, and the expected security from the Executor, also the Purchaser locks the reward tokens responsible of paying the gas: gasCost * gasAmount

.

The GasOrder should include such fields:

struct GasOrder { uint256 gasAmount; // The amount of Gas to book for future executions uint256 executionPeriodStart; // Start timestamp when it is possible to use the gas within the order uint256 executionPeriodDeadline; // End timestamp when it is possible to use the gas within the order uint256 executionWindow; // This variable defines a window, measured in blocks, within which a // transaction must be executed. This constraint is designed to optimize timing and prevent delays, while // also safeguarding against the exploitation of gas price fluctuations by malicious actors. TokenTransfer gasCost; // The cost of one Gas unit TokenTransfer guarantee; // The guarantee security required from the Executor }

Accepting order

Some Executor accept the conditions of the GasOrder, and locks the guarantee. The guarantee security is execpected to be proportional to the purchased amount of Gas.

Requesting transaction execution

When the GasOrder execution timeframe comes, the user might request transaction execution by signing the data structure with the transaction details.

Message { address from; uint256 nonce; uint256 gasOrder; // number of the employed order uint256 deadline; // deadline of the msg execution, should be within the range of order execution address to; // the contract which is being called uint256 gas; // gas limit to spend uint256 tips; // tips to the party which pushes the tx request onchain bytes data; // execution request details bytes signature; // the signature by the sender }

After the transaction is signed the hash of it should be published onchain. It might be done by transaction requester itself, or by anyone else. To incentives the posting the transaction request onchain the signer specifies the tips

. The tips represents the Gas within the GasOrder which will be burned and the respective share of reward will be directed to the transaction request submitter.

Executor takes the signed data and calls the to

contract from function within the protocol which executes the call with the data

from the Message

. Consecuently the call unlocks the share of the reward and the guarantee for the Executor.

If the transaction is not Executed because it is not profitable for the Executor than the Executor might be liquidated, it might be implemented in few ways, centralized and decentralized, lets review the decentralized version.

Decentralized liquidation logic

During the transaction request, the transaction hash is posted on-chain, also the signature and remaining Message

data posted as a calldata to be publicly available.

If the Executor fails to execute the transaction before the Message.deadline

, anyone can do so by providing the necessary data from the Message

before the deadline + CONSTANT_LIQUIDATION_TIME

. In return, the executor's guarantee is partially forfeited, and the Liquidator requester receives a reward.

Bottlenecks

Executor incentive

Executor incentives rely on adjusting GasCost to accommodate risk. As gas prices are unpredictable, Executors mitigate risks by charging extra. This flexible pricing model, determined by Executors, may evolve from sporadic agreements to a more standardized market, resulting in better price averages over time.

Gas consumption

The protocol's viability hinges on surpassing a certain threshold, as it necessitates gas for order publication, acceptance, signature verification, and transaction execution.

Split of liquidity

Tokenizing each Gas order is straightforward, yet trading shares between orders poses a challenge due to their differing parameters. While securitization of long-term orders seems a plausible solution, preventing liquidity fragmentation remains elusive at present.

Lock of guarantee

Executors must lock guarantees to deter liquidation risks, yet this restricts their flexibility. The locked guarantee could otherwise be utilized for liquidity elsewhere to generate yield. One potential solution is to lock tokens which represent shares in farming pools, enabling yield generation while locked. However, this introduces additional risks for gas Purchasers parties, as farming protocols entail additional security assumptions and associated risks.

P.S. I'm really interested to get the feedback on the proposed mechanism