Scaling Decentralized Exchanges through Transaction Aggregation

Paul Etscheit[1]

Information Systems Engineering
TU Berlin, Germany
p.etscheit@tu.berlin

1 Context

In recent years, the popularity of ERC-20 tokens has started a growing trend of developing smart-contract based, decentralized trading on the Ethereum blockchain. While centralized exchanges have existed for quite a while, they were the only choice available to people looking to trade ERC-20 tokens. While centralized exchanges (e.g. Binance) are quite sophisticated and can handle large trading volumes, a core problem is, that user funds are stored custodial on the exchange and are a target for hackers, looking to drain the exchange's wallets. While proper security measures can reduce the risk of an exchange getting hacked, it tends to happen quite frequently in the space.

The most popular decentralized exchange today is called Uniswap [5], which is running as a smart-contract on the Ethereum blockchain. When trading on Uniswap, the entire trade is done in a non-custodial manner. In practice, this means, that a user will pay for the trade from their local wallet, and receive the traded ERC-20 token to the same wallet. This operation is atomic, so a user never loses custody of their coins. Since the wallet is running locally on client-side hardware, the risk of being targeted by hackers is substantiality lower, compared to wallets of centralized exchanges. While it increases the security of funds, increased trading activity on Uniswap causes gas fees to rise substantially, which is not only a problem to Uniswap traders but all network participants who want to make transactions on the Ethereum blockchain.

2 Problem

The recent months have shown a large increase in trading activity on Uniswap, reaching close to one billion dollars worth of tokens being traded per day in early September [4]. A typical Uniswap trade uses around 125k gas, which is around 1% of the current block gas limit (around 12.5 million [2]) and results in a block only being able to include a maximum of around 100 trade transactions, or around 7.7 trades per seconds (block time of 13 seconds on average [1]). Keeping in mind, that these numbers would completely block the network for any other dapp or network participant, it becomes quite obvious, that scaling of some sort must be achieved. On the 2. September of this year, the average

gas price reached 480 Gwei [3]. This correlates closely with the peak trading activity on Uniswap, and resulted in a normal trade, costing around 0.06 ETH in transaction fees, or around $27 at current prices (06.11.2020). While this is an obstacle for efficient token trading, it also harms other dapps, that depend on the gas price being somewhat more reasonable, so reducing these fees is beneficial to all network participants. While the gas price has stabilized to more normal levels in recent weeks, the core problem of scaling the network persists, and peaks like in September are to be expected again in the future.

3 Solution

A potential solution lies in the utilization of zkSNARKS [7], a specific type of zero-knowledge proof, for achieving a higher transaction throughput per block, while not adding any kind of trust or data availability assumptions. These proofs can be used to process a computationally expensive operation off-chain, and verify the validity of the result on-chain. zkSNARKS are favorable in the context of any kind of blockchain application, as their proof size is constant, no matter how complex the computation being proven is. This, among other things, promises to scale Ethereum's transaction throughput, with a process known as zk-rollup [8]. Zk-rollups essentially enable the bundling of user transactions off-chain and verify the correctness on-chain with a zkSNARK proof. By doing that, hundreds of transactions can be bundled into a single one, which greatly reduces gas consumption and in effect strain on the Ethereum network. While this has already been developed for Ether or ERC-20 tokens (further known as assets) transfers, the novelty lies in utilizing this process around an entire dapp, and not only the transfer of assets. This implementation will be focused on integrating zk-rollups with the uniswap contracts, however, the general approach is not limited to Uniswap and can be utilized with other contracts as well.

4 Implementation

The core idea behind zk-rollup, is that users deposit assets into a smart-contract, which in turn tracks the balances of a user off-chain in a merkle tree by storing the merkle trees root. When a user makes a transfer transaction within the zk-rollup system, the actual assets aren't moved, but instead, the balances in the merkle tree are updated and the correctness of that update is verified with a zkSNARK proof. By doing this, we're able to bundle a large number of transactions into a single one, thereby saving gas. Since a user doesn't have the actual assets in their wallet, but rather access to that balance via signature, a withdraw function can be used to get the actual assets back from the smart-contract, into the user's wallet.

This mechanism can also be used for bundling a large number of Uniswap trades into a single on-chain trade function call. As explained, users must first deposit their assets into the smart-contract. A user can then call the trade function of the UI for a specific asset pair, which is broadcasted to the off-chain entity, the

relayer, and contains the relevant information for the trade (assetFrom, assetTo, amount, tradePool) and a valid signature (this process is completely off-chain). A trade pool is used to gather user's trades and and will have them executed at a specific block number. When the target block number is reached, the relayer checks the signatures of all trades, dismissing the ones that are invalid, and bundles them. Since all pairs can only be traded bidirectional (sell A for B, buy A for B), the first step is to offset these trades internally, resulting in one direction (buy or sell) to equal zero. As a next step, the remaining asset needed for all trades to be successful is bought with a single Uniswap trade transaction by our smart-contract. Once that is completed, the new balances of all users that participated are updated in the merkle tree, and the operation verified via zkSNARK proof. While verifying we must pass the individual trades as public input to the verify function. This is needed to make sure the relayer has worked with the correct inputs and also writes the different trades on the blockchain. By doing this, we can build the system without having any kind of data availability issues, since all data needed to recreate the merkle tree can be found on-chain. Users now have access to their traded assets and are free to withdraw them or keep them in the smart-contract and sell at a later time.

For the zkSNARKS proofs and verification, I will choose ZoKrates [6], a zkSNARKS toolbox that includes Ethereum verification contracts. The fact that the verification contracts are generated is convenient and I have some experience working with it. Other than that, I will use web3 in some form and solidity for writing the smart-contracts. While this is not a complete list of tools and frameworks that will be used, I'm quite certain that these tools will be used.

5 Evaluation

Since this system will have a fixed amount of gas overhead, the cost per trade depends on the number of users that are part of a trade pool. The more users are in a trade pool, the lower the fees per user will be. Since the gas cost for a normal Uniswap trade is constant, this metric can be benchmarked nicely, simulating users buying and selling and then comparing the cost per trade. Since calculating a zkSNARK proof is very resource-intensive, the proving time is also an interesting metric for this system since it will somewhat limit the frequency of trade pool executions and general transaction throughput of the system. It will be interesting to benchmark these numbers on different types of hardware with a varying amount of transaction in a trade pool. I believe these two main metrics will give a good indication of how well the system performs and what benefits can be expected of it.

References

1. Ethereum average blocktime, https://etherscan.io/chart/blocktime
2. Ethereum block gas limit, https://etherscan.io/chart/gaslimit
3. Ethereum gas price, https://etherscan.io/chart/gasprice

4. Uniswap protocol analytics, https://info.uniswap.org/home
5. Adams, H., Zinsmeister, N., Robinson, D.: Uniswap v2 core. URl: https://uniswap.org/whitepaper.pdf (2020)
6. Eberhardt, J.: Zokrates—a toolbox for zksnarks on ethereum. Ethereum Foundation [online]: https://ethereumfoundation.org/devcon3/sessions/verifiable-off-chaincomputation-for-smart-contracts/,(accessed on 10-02-2018) (2017)
7. Foundation, E.: zksnarks in a nutshell (Dec 2016), https://blog.ethereum.org/2016/12/05/zksnarks-in-a-nutshell/
8. Vbuterin: On-chain scaling to potentially 500 tx/sec through mass tx validation (Sep 2018), https://ethresear.ch/t/on-chain-scaling-to-potentially-500-tx-sec-through-mass-tx-validation/3477