Rubicon v1[1]

Benjamin Hughes, Forrest Norwood
contact@rubicon.finance
March 19, 2022

Abstract

In this paper, we introduce Rubicon v1, an order book protocol for Ethereum, and by extension an order book protocol for the world.

Background

An exchange is the most important venue in a given financial system; without a common venue for liquidity, markets are inefficient. Given their importance, the ideal exchange is fully transparent, has no downtime, and is accessible to anyone wanting to participate in the most basic transaction in finance: a trade.

Today, order books[2] are the most popular form of exchange. An order book is a list of buy and sell orders for an asset organized by price level; it is the simplest way to reflect supply and demand for assets. With limit orders, one has the ability to choose the price, size, and direction of a trade. Price discovery for assets in financial markets is a dynamic process, and order books give traders unlimited flexibility to adjust their positions to reflect new information in real-time.

In 2008 Bitcoin[3] began a revolution, showing the world an alternative to trust-based payment systems. Ethereum[4] extended this underlying blockchain technology to any possible application by giving developers the ability to write smart contracts. Modern finance is infested with middlemen, so financial applications were a natural fit for many of the first decentralized apps.

As decentralized finance (DeFi) applications began to flourish on Ethereum, there were several attempts to implement order books to serve these nascent markets. These attempts failed, mostly due to the high transaction costs and long confirmation times on the base layer.

The shortcomings of order books in these environments led to most decentralized exchanges (DEXs) adopting the Automated Market-Maker (AMM) model, which utilizes liquidity pool reserves to algorithmically determine price across a constant function. This model can

operate within the constraints of Ethereum's base layer, but in all their forms AMMs rely on a class of sophisticated, highly-active traders to keep their liquidity pools in line with market prices. Their inability to adjust to the dynamic price discovery process, combined with necessary arbitrage flow, means AMMs often produce poor returns[5] for liquidity providers.

Notably, some projects chose to escape the constraints of the blockchain by operating off-chain order books or matching engines which settle trades on-chain. In doing so, they also escape the security, openness, and composability that make blockchains valuable in the first place.

An off-chain order book creates a single point of failure, making it incapable of realizing the full potential of a decentralized exchange. An order book protocol is robust and can settle the world's financial markets, a server privately operated by a single entity cannot. Luckily, there exist scaling alternatives other than outright centralization!

Years of research have gone into scaling Ethereum to help the blockchain fulfill its potential as a world computer. The most promising solutions to come out of this work are rollups[6]. In a rollup, transactions are executed by an off-chain operator, and the resulting data and state are batched back to the primary blockchain. Fraud proofs or validity proofs are used to ensure the off-chain operator submits only valid data, so it is often said that rollups inherit the full security guarantees of Ethereum.

Rollups are a superior scaling solution because they can bring almost unlimited computing power and storage to decentralized applications without compromising on security. Dial-up internet was revolutionary when it was first introduced, however, less than a decade later it was completely obsolete due to the emergence of commercial broadband. In the same vein, existing applications are over-optimized for the conditions of Ethereum's base layer, even as there is community consensus[7] that most activity will soon be on rollups.

Both in and out of crypto markets, order books trend towards centralization. That is, they are typically reliant on a few professional trading firms to provide liquidity for traders. In legacy markets and centralized crypto exchanges, some trading firms are covertly given priority and benefits that are not available to the average trader. When a small number of firms have significant control over an exchange, perverse incentives arise[8].

Every trade on an order book has two sides: a maker and a taker. Legacy incumbents are happy to give everyone the ability to be a taker, charging them premiums in the process. However, the ability to be a maker is closely guarded, and although crypto markets are more transparent than traditional ones, providing liquidity is still not accessible to the average market participant.

We believe providing liquidity is a financial opportunity that should be open to anyone. Instead of bending the knee to proprietary capital, we sought to build an order book protocol that levels the playing field for all participants. Ethereum's order books should give priority to no one.

Rubicon v1

Rubicon v1 is an order book protocol with native liquidity pools. Its cornerstone is the RubiconMarket contract, which implements an open order book for the peer-to-peer trading of ERC-20 tokens. The contract holds all outstanding offers on the order books in escrow.

The simplest way to submit an order to RubiconMarket is to call the offer() function. The function requires a pay (sell) amount, an address for the token being paid (sold), a buy amount, an address for the token being bought, and the position to insert the offer (set to zero by default). Orders submitted using offer() will match against all other orders in the order book, and can be partially filled, leaving a remnant amount on the order book. Orders that cross the spread are matched on-chain in real-time.

Every order on RubiconMarket has a unique order ID that can be used to view that order's details or to interact with the order directly. For instance, the cancel() function uses an order ID to cancel a specific order in the book. Likewise, the buy() function uses the order ID to fill a specific order on the book.

An order submitted using the sellAllAmount() function specifies a pay (sell) amount, an address for the token being paid (sold), an address for the token being bought, and a minimum fill amount. The fill amount (execution) for the trade can be better than the minimum amount, but if it is below the transaction will revert. This provides a convenient way to sell an ERC-20 as a market order.

Similarly, an order submitted by calling buyAllAmount() specifies a buy amount, an address for the token being bought, an address for the token being paid (sold), and a maximum fill amount. The transaction will revert if the fill amount is greater than the specified maximum. This function represents a market buy on the ERC-20 pair of the user's choice.

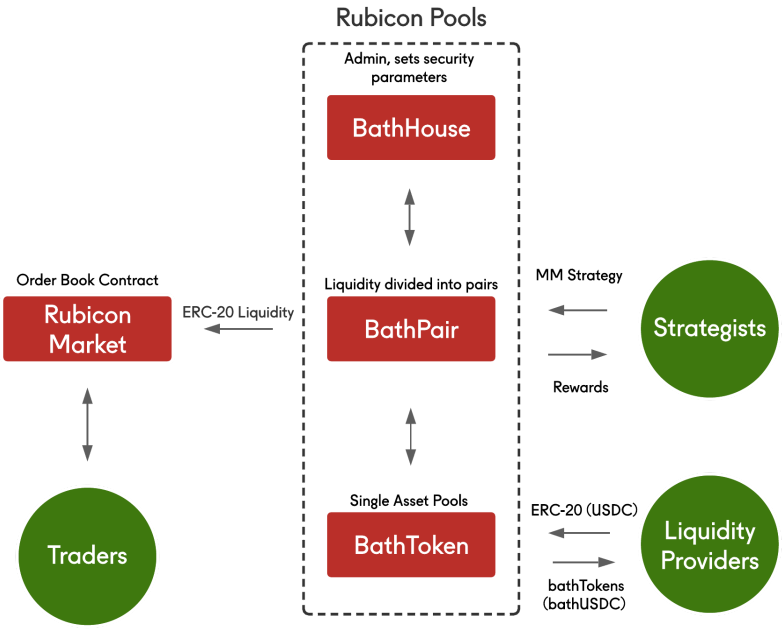

Figure 1

Rubicon Pools are the native liquidity pools for the Rubicon order books. Passive liquidity providers (LPs) deposit tokens into single asset pools, where assets are then deployed by active liquidity managers (strategists) to make markets on the order books. Yield from market-making is passed to the pool liquidity providers, while strategists earn a performance fee from their filled orders.

The BathToken contract implements a liquidity pool for an ERC-20 token. A liquidity provider deposits tokens into a contract, which mints them a corresponding amount of bathTokens. Each bathToken represents one share of the claim on the underlying ERC-20 tokens; these shares are minted and redeemed according to the ratio of bathToken totalSupply() to the pool's underlyingBalance() of ERC-20s - this represents the virtual price of a bathToken. We use the prefix "bath-" to denote LP tokens in Rubicon Pools e.g. if someone deposits USDC, they will receive bathUSDC. Conversely, to withdraw from the pool an LP burns their bathTokens and receives their share of tokens from the underlying pool.

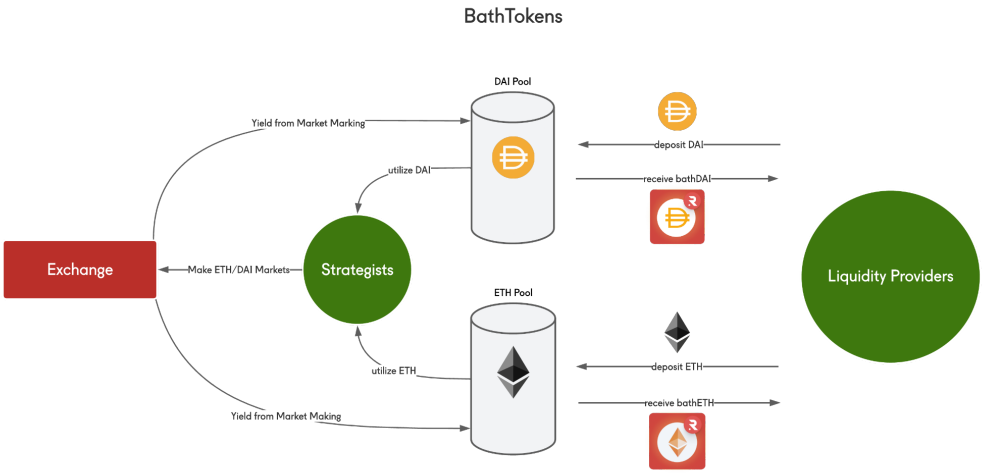

Figure 2

As seen in **Figure 2**, a liquidity provider is essentially buying a share in the pool, in return, they are minted bathTokens that represent their proportional ownership of the pool's underlying assets.

BathTokens can be redeemed for the underlying tokens at any time. There is a fee paid by LPs upon withdrawing from a pool. This fee goes to the pool itself, rewarding the remaining liquidity providers, and is in place to prevent bad actors from attempting to manipulate bathToken virtual prices.

BathTokens are yield-bearing order book liquidity for the underlying token. We expect bathTokens to be used not only on Rubicon but across a number of new and existing applications within DeFi.

Rubicon v1 uses single-asset pools so that liquidity is not "siloed" into one trading pair. With each asset having its own pool, liquidity can actively adjust to demand, flowing between

pairs to achieve higher utilization. This is handled by strategists, so liquidity providers do not need to actively manage their positions.

The single-asset design boosts both the efficiency of liquidity management and the user experience of Rubicon Pools. The typical passive liquidity provider does not want the responsibility of managing their positions, they want optimized yield on their assets. On the other hand, high-frequency traders and market makers want access to more liquidity to implement their strategies and levered upside via performance fees. By outlining clear roles for both parties and aligning their incentives, Rubicon v1 harmonizes active and passive liquidity, inheriting the benefits of both.

The BathPair contract is the primary Rubicon v1 touchpoint for strategists, the active liquidity managers in the Pools system. The contract enforces security parameters on strategist trades and tracks their performance.

By calling the placeMarketMakingTrades() function, strategists can place bids and/or asks on the order books using liquidity from the pools. In Rubicon v1, strategists can submit maker or taker trades with pool liquidity. In practice, they rarely take liquidity off the order books unless there is a clear incentive to do so. All profits from arbitrage or other alpha opportunities within a network are passed back to LPs less the strategist performance fee.

BathPair also tracks strategists' outstanding orders and logs their performance for rewards. They can claim their rewards for a trading pair by calling the strategistBootyClaim() function on the contract.

BathPair also contains the logic for system-wide inventory management. When a trader matches with a strategist order on RubiconMarket, the filled order is sent to the pool which originated the liquidity for that trade.

The liquidity pools accrue tokens that are not their native token in the form of fills. To realize this yield for the pool liquidity providers, the fills must be swapped back into the pool's native token. The first option to realize this yield is to rebalance fills between liquidity pools internally. This allows both liquidity pools to profit from the spreads they used on market-making orders; the aspiration for strategists is to balance filled orders on each side of a trading pair so pool rebalancing can stay internal.

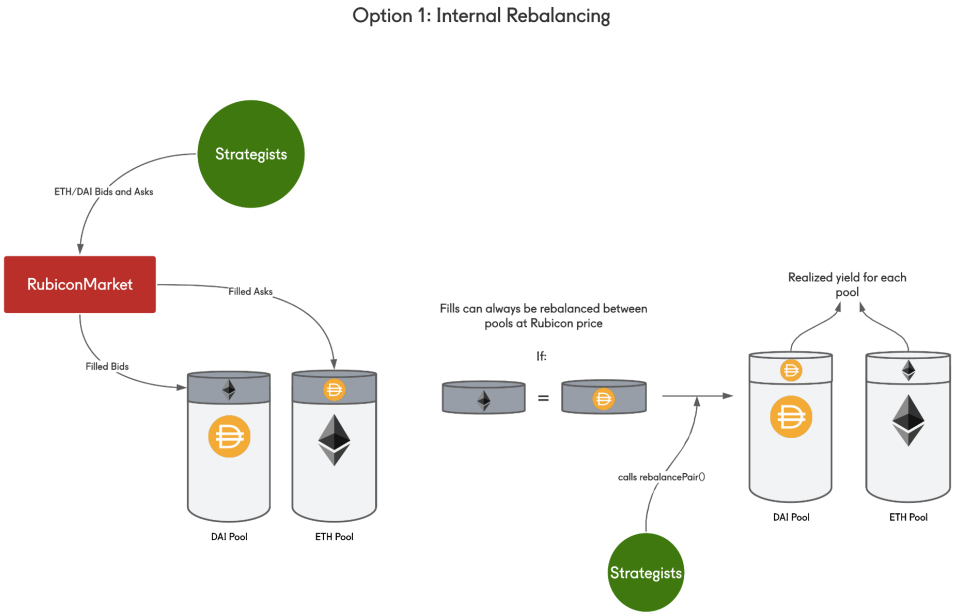

Figure 3

In the example in **Figure 3**, the ETH pool is accruing DAI fills and the DAI pool is accruing ETH fills. These fill amounts can be swapped between pools at a set price by strategists calling the rebalancePair() function on BathPair, realizing the yield for both pools. In v1 the strategist rebalances internally at market rate to ensure matched fills.

If the fill amounts cannot be swapped internally, strategists can and should use an external liquidity venue like an AMM to rebalance fills and offload risk on an external pool.

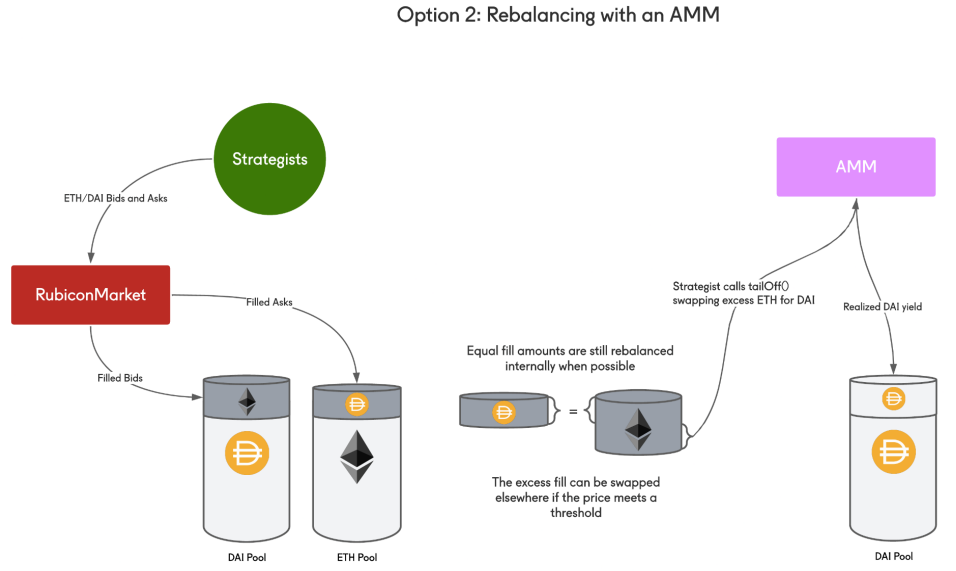

Figure 4

In the example in **Figure 4**, there are fill amounts on the pools that cannot be swapped internally without one of the pools realizing a loss. To swap the fill amount back to the native

asset and realize yield for the pool, a strategist can call the tailOff() function, which allows for strategists to make swaps with fills on external venues thus returning assets and profits to LPs.

The BathHouse contract acts as the system admin and provides key high-level functions for Rubicon Pools. It contains the setter functions for all security and system parameters within the Pools contracts. These parameters are variables that are subject to change and will be optimized as we study and monitor the health of the Rubicon Pools system.

BathHouse sets the following security parameters to protect Rubicon Pools LPs:

- ReserveRatio ensures that some amount of pool liquidity is present in the contract at all times. This protects the pools from overutilization by strategists and ensures that a portion of the underlying pool assets are liquid so LPs can withdraw. The ReserveRatio is initially set to 50, meaning 50% of a liquidity pool's assets must remain in the pool at all times.

BathHouse also sets the following system parameters:

- BPSToStrats is the performance fee for strategists in basis points. It is currently set to 20, so a strategist earns 0.2% from orders they placed that are filled as soon as fill is managed and profits in the form of the pool's underlyingERC20() are returned to LPs.

- Market is the current address of the RubiconMarket contract. Rubicon pools can only deploy capital into RubiconMarket for market-making purposes (conversely, fills can be managed arbitrarily for NPV+ opportunities within the network).

- FeeBPS is the fee for withdrawing from a liquidity pool in basis points. The fee is currently set to zero but could change in the near future. These fees are paid back to the pool and reward bathToken holders.

- FeeTo is the destination address for the withdrawal fee. These fees go to the pool, so it is set to the BathToken contract address for a given pool.

- ApprovedStrategists adds an address that can be a strategist and utilize pool liquidity. This is currently restricted to addresses controlled by our team as we develop this novel liquidity mechanism further.

Once BathHouse initializes the Pools contracts and sets these parameters, liquidity providers can begin depositing assets and strategists can begin deploying pool liquidity onto the order books.

BathHouse also houses the functionality for pool creation in Rubicon v1. To create a new liquidity pool, call the openBathTokenSpawnAndSignal() function and deposit tokens into the new pool, as well as other tokens into an existing liquidity pool. This second deposit serves as a signal to strategists of what the first trading pair for the new pool should be, and gives strategists a benchmark price to build the initial order book.

In review, Rubicon v1 is an order book protocol for ERC-20 tokens that offers native

liquidity pools. The protocol is built on rollups to minimize costs and confirmation times while maintaining Ethereum-level security. The protocol democratizes market-making through Rubicon Pools, a system of contracts that outlines explicit roles for both active market-makers and passive liquidity while aligning their interests.

Conclusion

We believe that in the near future, the majority of the world's financial activity will be settled on Ethereum. In order to achieve the scale necessary for global adoption, most of this activity will be executed on rollups.

Ethereum needs a core order book protocol to surpass legacy financial markets. The logic for this protocol must be fully on-chain, where its markets are not reliant on a single service provider and cannot be manipulated by a single entity. Additionally, the exchange must not rely on a small number of trading firms for liquidity, this ensures incentives are aligned with every trader, not only institutional ones.

Rubicon v1 is an order book protocol built to fulfill this vision. It relies only on the security of the underlying blockchain to stay online. Rubicon Pools democratizes the liquidity landscape, unites active and passive liquidity providers, and balances their incentives so they serve each others' interests.

A core global open order book protocol with no barriers to entry will be the world's most efficient financial market. It allows for a perfect-information, scalable, and trustless platform in which everyone in the world has root access; discrimination is programmatically impossible. Information from every corner of the globe will be reflected through prices, real-time, on a common exchange venue. If Ethereum is a World Computer, Rubicon is a World Order Book.

Future Works

The core order book contract in Rubicon v1 builds upon the open-source work of MakerDAO[9]. While the current version is robust after having spent years on mainnet, a future version could be more gas-optimized, and it is likely that the matching engine could be more efficient.

Rollups have unique methods for calculating transaction costs, so future upgrades could optimize the Rubicon v1 contracts' performance on a specific network. We also anticipate other novel features and benefits from these networks. For example, on some zero-knowledge rollups, it is conceivable to have gasless limit orders for traders with no compromises on security.

The primary focus for future versions of the protocol will be the decentralization of market-making and the strategist's role in Rubicon Pools. This role is currently permissioned; our team acts as the sole strategist while we refine this new system to prohibit malicious liquidity management and optimize for future success.

The first application of Rubicon v1 will be spot crypto markets. Importantly, the applications

for a permissionless order book protocol are unlimited; it can support markets for any tokenized asset. Future improvements could introduce more products and new types of markets.

Maximal Extractable Value[10] (MEV) is the maximum amount of value that can be extracted through reordering, censoring, or inserting transactions within blocks being produced. MEV on layer 2 networks is an active area of research, and many L2 teams have not finalized their block production and sequencing mechanism. Future versions of the protocol could use private mempools and/or other solutions to protect Rubicon trades from value extraction.

References

[1] Hughes, Benjamin. "Rubicon v1.0.0". GitHub. March 2022. https://github.com/RubiconDeFi/rubicon-protocol-v1

[2] "Order book". Wikipedia. February 2022. https://en.wikipedia.org/wiki/Order_book

[3] Nakamato, Satoshi. "Bitcoin: A Peer-to-Peer Electronic Cash System". October 2008. https://bitcoin.org/bitcoin.pdf

[4] Buterin, Vitalik. "Ethereum Whitepaper". Ethereum Foundation. 2013. https://ethereum.org/en/whitepaper/

[5] Stefan Loesch, Nate Hindman, Mark B Richardson, Nicholas Welch. "Impermanent Loss in Uniswap V3". arXiv. November 2021. https://arxiv.org/abs/2111.09192

[6] Wackerow, Paul. "Rollups". Ethereum Foundation. Last modified January 2022. https://ethereum.org/en/developers/docs/scaling/layer-2-rollups/

[7] Buterin, Vitalik. "A rollup-centric ethereum roadmap". Ethereum Magicians. October 2020. https://ethereum-magicians.org/t/a-rollup-centric-ethereum-roadmap/4698

[8] Securities Exchange Commission. "Citadel Securities Paying $22 Million for Misleading Clients About Pricing Trades". November 2017. https://www.sec.gov/news/pressrelease/2017-11.html

[9] https://github.com/OasisDEX/oasis

[10] Maximal Extractable Value. https://docs.flashbots.net/new-to-mev

Acknowledgments

Thank you to Denver Baumgartner for his thoughts, criticisms, and suggestions while reading drafts of this paper.