Conveyor Finance Whitepaper

Conveyor Labs
contact@conveyorlabs.org
https://conveyor.finance

June 7, 2022

Abstract

Conveyor Labs created the first and currently only available protocol that enables fully decentralized, trustless, and non-custodial swap automation for DeFi traders. Through a pairing of an open-source, distributed, off-chain network of "Beacons" and on-chain logic and validation, our protocols enable users to leverage buy sell limits, stop losses, and other trading strategies between two tokens. A user of our system is able to confirm desired swap conditions through the Conveyor.Finance frontend, which immutably stores their sell conditions on-chain, while also authorizing the protocol to spend their defined amount tokens held in the users wallet when the on-chain market conditions are met.

1 Introduction

This white paper is designed to be straightforward and easy to read with the purpose of having an individual fully understand how the Conveyor ecosystem works after reading. This document describes the completed version of Conveyor at the end of the roadmap with all upgrades. However, there will be a section at the end explaining the current state of development as well as the roadmap for Conveyor.

2 Overview

Conveyor.Finance is a decentralized application that acts as a sidecar to your cryptocurrency wallet, allowing you to automate your wallets' token swaps 24/7. Conveyor has cross chain compatibility, and will have eventual support for any EVM compatible blockchain, side-chain or Layer 2 and is compatible with any cryptocurrency wallet that uses an Elliptic Curve Digital Signature Algorithm (ECDSA) to generate public/private key pairs. There are two major parts to the Conveyor ecosystem. The Beacon application and the on-chain smart contract. These two components work together to identify market conditions, validate on-chain LP values, and execute specified swaps autonomously for users of the system. Conveyor takes the centralization or custodial requirements out of swapping tokens so that institutional traders, investment DAOs, token developer teams, and retail traders don't need to constantly watch charts in anticipation of manually swapping when their desired market conditions are met.

3 Transactions

The transaction life cycle goes through various steps during order placement, spend authorization, off-chain listening of on-chain state, off-chain interaction with the protocol, on-chain market validation, and finally swap execution. All of these steps are individually necessary in order to maintain a fully decentralized system without reliance on the Conveyor infrastructure.

3.1 Order Placement

First, a user navigates to Conveyor.Finance website, which has an interface similar to centralized exchanges. They are presented a frontend that allows them to directly connect their web3 wallet like Metamask, Trust Wallet, Safepal, etc... After entering their desired swap conditions (eg. Stop-loss at a certain USDC value - or - a buy limit to swap $TokenA \rightarrow TokenB$ when a specific ratio is met), the user signs a transaction that stores their desired swap conditions on-chain.

3.2 Spend Authorization

Upon order placement, the contract is given authorization to spend the defined amount of tokens held in the user wallet to fulfill the swap when the market conditions are met.

eg. If a user places a stop-loss to sell 1 ETH for USDC when X price point is met, the contract is given spend authorization to spend 1 ETH in a users wallet.

When an order is either fulfilled or canceled, the spend authorization is decremented that order value.

eg. If a user has two limit orders that each swap 1 ETH, there is a spend authorization of 2 ETH given to the contract. Canceling or fulfilling one of these orders decrements the spend authorization by 1 ETH, resulting in the contract only authorized to spend 1 ETH for the remaining order.

3.3 Off-Chain Listening of On-Chain State

Conveyors distributed Beacon application is able to read the pending users' orders on-chain and simulate the various trade conditions off-chain. The information they can see is the users public wallet, their order ID, and the desired trade conditions for that specific order.

3.4 Off-Chain Interaction with Protocol

When a Beacon locally simulates a trading scenario that meets the users specific trade conditions, it can determine when interaction with the protocol would pass all logic and validation checks. When the on-chain market conditions are validated by the Beacon, the Beacon interacts with the public functions of the contract by entering the required swap criteria and spends the gas needed to run the logic for completing the users' transaction.

3.5 On-Chain Market Validation

Security of users' funds requires that the Beacon cannot be trusted to act faithfully when entering swap values criteria. After a Beacon interacts with the contract and before a transaction can be successfully completed, the contract trustlessly checks multiple LP reserve values in order to determine

the price of the assets it's swapping to/from. If the price determined on-chain does not meet the specified users' swap values, the transaction will immediately revert before any swap has been made. The Beacon forfeits the gas cost if this were to happen, which creates a financial disincentive for Beacons to pass in faulty/incorrect data or blindly spam the contract in the hopes of a transaction succeeding.

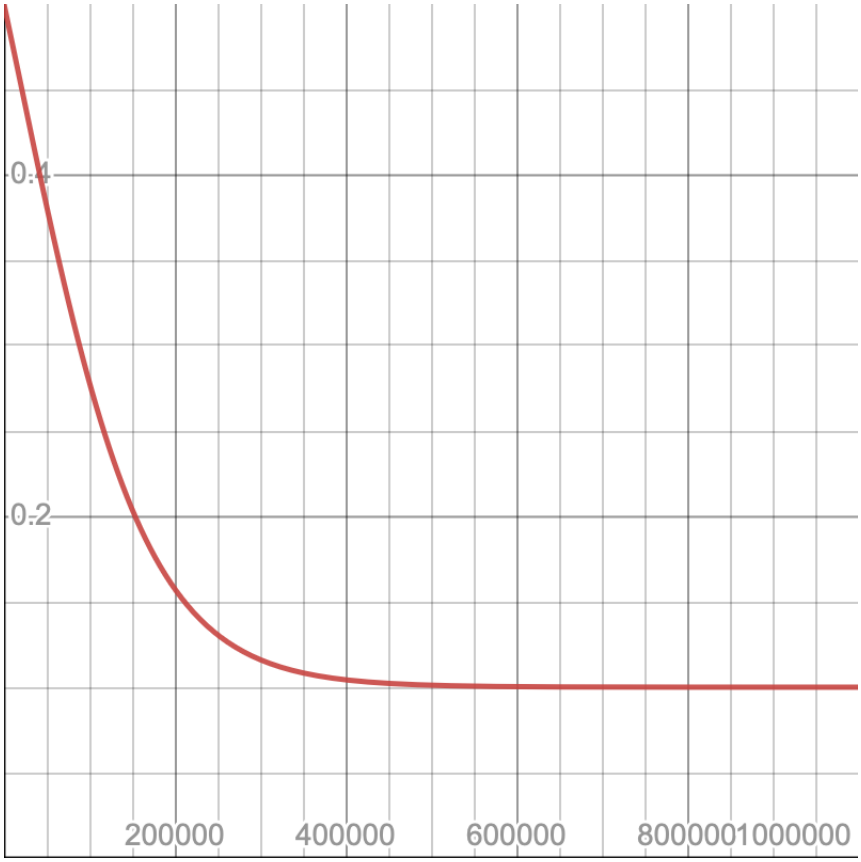
3.6 Swap Execution

When all on-chain checks have passed validation, the contract generates a transaction to swap the approved tokens within the users wallet by leveraging the most favorable trade conditions across multiple DEX LP's. When the most favorable swap pathing has been assessed, the contract does a 'transferFrom' the users wallet and interacts with the appropriate LP's to swap the 'From' token to the 'To' token. This process requires no signatures or manual process from the end user due to the spend authorization the contract has of the users' tokens in their wallet.

4 Fees

A small automation fee is removed mid-swap based on the transactional value of the swap (between $0.5 - 0.1$ percent based on lower vs higher value trades, respectively) which is split between the beacon that facilitated the transaction, and Conveyor. The fee follows a logistic curve rather than a 'tiered' value fee. Our modified logistic curve is as follows

$$f(x) = \frac{0.9}{1.25 + e^{\frac{x}{75000}}} + 0.1 \tag{1}$$



This fee structure reduces on-chain computation, resulting in cheaper overall fees. This also maintains a scalable structure that results in transacting a token value equaling 101 USDC incurs a cheaper transaction fee than a similar transactional value equaling 100 USDC. The Beacon fee + Conveyor fee will always total to the curved line of the above graph. The reasoning behind using a logistic curve, rather than a straight line, is to encourage the user to place higher value orders instead of multiple smaller trades, which is less costly for the user in terms of gas, and doesn't require a modifier when the straight line intersects the 0.1 percent fee.

The Conveyor vs Beacon fee breaks down to a 60/40 split, respectively, for lower value transactions. For higher value transaction, the ratio inverts, where the Beacon earns closer to 60 percent of the reward, with Conveyor receiving closer to 40 percent. This provides a growth mechanism for contin-

ued development on the Conveyor ecosystem with lower transactions, with higher value trades creating a financial incentive for the Beacon network to expand and further decentralize.

5 Risks/Mitigation

5.1 Flash Loan Attacks

Flash loan attacks on our user base pose a significant threat to the protocol by bad actors in the Beacon network. Traditional DeFi automation networks such as The Gelato Network have an internal executor network to call the execution functions of the on-chain logic. The Conveyor Beacon network is completely democratized, allowing anyone to spin up a beacon client and start executing incoming orders in the queue. The reason why most other well-known DeFi automation protocols keep their executor network internal is because of the risk of bad actors in the Beacon network. Since the Beacon network will be receiving rewards by executing orders stored in the contract, it is possible for a Beacon to manipulate the spot price in a liquidity pool to drive the price to the execution trigger for a batch of transactions to execute, in such cases where the execution reward is significantly large.

The Conveyor protocol has deeply analyzed this problem, and mathematically formulated a model on when such an attack will be profitable. Our protocol's Beacon reward is analytically determined with flash loan attacks in mind and is hard-capped at the exact value that would make such an attack profitable. Meaning, a Beacon could take out a flash loan to manipulate the price of an LP to the execution trigger for a batch of transactions and at best will be breaking even - the gas accumulated in the transaction. Alternatively, the Beacon could simply play by the rules and reap the max execution reward, receiving a net-positive return.

The overall philosophy of the Conveyor protocol has been to mechanistically design the system in such a way that the most profitable way to use the platform is also the correct way to use the platform, and flash loan attack prevention has been mechanistically designed into the system to be unprofitable so attempting such attacks will invariably lose money for bad actors in the network.

5.2 Contract Security

The conveyor contract has been designed with security as the top priority. Our engineers have designed a custom EOA library to further strengthen the security of our execution function from external contract calls. This library protects against sophisticated ghost contract calls which several modern protocols which only check the code size of the caller are still vulnerable to.

The Conveyor contract will be audited by QuantStamp prior to our protocol launch to ensure a deep level of security and professional oversight to our user base.

Although we have made deep mechanistic design decisions to prevent flash loan attacks on the liquidity pools to trigger order executions. We further implement price oracles as secondary price indicators alongside the liquidity pool spot prices based on reserve size. This is to have redundancy in our execution price triggers in order to prevent execution on a manipulated spot price in a liquidity pool.

6 Gas Credits

For an order to execute, the Beacon must call the fufillOrder() function on the Conveyor contract. The Beacon spends the gas necessary to execute the order, but what happens if the cost to execute the order is higher than the profit from the limit order itself? This is where gas credits come in. Gas credits are an important concept to the Conveyor ecosystem, ensuring that Beacon runners are incentivized to execute limit orders as fast as possible.

6.1 Gas Credits Explained

When a user first places an order, they are required to deposit gas credits to cover the cost the Beacon incurs when executing the order. Simply put, a user stores a small amount of ETH in the Limit Order contract to pay the Beacon for order execution. A user must add a minimum amount of gas

credits, dependent on the current fair gas price at order placement to ensure order completion. In the case where gas price increases and the user does not have enough gas credits to cover order execution, the Beacon would not have an incentive to fulfill the order, thus risking order cancellation. It is recommended that a user deposits enough gas credits to account for gas price fluctuation.

All gas credits are accounted for within the gasCredits mapping stored in the Limit Order contract, keeping track of how many gas credits a user has at any time. Users can add to their gas credit balance whenever they would like and can withdraw all gas credits as long as they have no pending orders. If a user has a pending order and wants to withdraw their credits, they must first cancel the order, and then they will be able to withdraw their full balance.

It is recommended that a user deposits at least 3x the current gas credits requirement in order to sufficiently cover the potential for volatility in gas prices. With this is mind, a user should always over estimate the amount of gas credits you put in to ensure coverage. The conveyor front-end web interface will always keep track of the gas prices in real-time and notify the user if their current gas credit balance is below the threshold for execution, however the user must maintain their gas credit balance above the necessary threshold.

6.2 Gas Credits Strengthen The Ecosystem

In order to maintain full decentralization and trustlessness within the Conveyor ecosystem, all values must be verified on-chain from execution price to gas price. This ensures that a Beacon can not input arbitrary values and keeps the system fully trustless. In order to execute this logic the Beacon must spend gas, incurring a cost to interact with the contract. In the event that the profit gained from order execution does not cover the cost of order execution, the Beacon would be losing money to interact with the contract. This is where gas credits come in. Gas credits ensure that the order execution is profitable for the Beacon, meaning that the Beacon is assured an incentive to call the contract to execute the order. Since there is an incentive to fulfill the order, Beacons will race and compete to execute the order which benefits

the user by ensuring order fills as fast as possible.

Gas credits also help to reduce contract storage bloat, which makes reading and writing from the contract's storage more expensive. Since there is a minimum gas credit upon each order, this discourages contract spamming. Additionally, due to order cancelation upon a user's gas credits falling below the necessary threshold for Beacon execution, the contract storage is able to stay lean.

6.3 How Is Gas Price Estimated?

When determining how many gas credits should be spent for order execution, gas price is verified on-chain to ensure a fair market price for execution.

Initially Conveyor will be using ChainLink's Fast Gas Oracle, which updates gas price whenever gas fluctuates 25 percent from its current price, or after 7200 seconds has passed since the previous update.

While the initial gas oracle will be Chainlink, Conveyor will be developing a fully decentralized, trustless gas oracle with block to block precision, cheaper execution costs, and fees paid in gas Ether instead of a non-native token.

7 Roadmap

Conveyor Labs, the development team behind Conveyor.Finance, plans to roll out a few major updates that will help to build the Conveyor ecosystem and improve the existing codebase. We are always actively listening to the community and plan to roll out more updates as the ecosystem grows. Below are a few updates that will be implemented in 2022 2023.

- Take-profit, Dollar Cost Average (DCA) Strategies, Grid Trading, and Leveraged trading.

- Launch on additional EVM compatible L1's/L2's like Arbitrum, Near, Avalanche, Gnosis Chain, and Fantom, having compatibility with major DEX LP's on each chain.
- Partnerships/integrations with low-code/no-code DAPP builders for extended reach and exposure.
- Create an OTC platform for non-custodial, protocol-based transfers between two or more parties on all EVM chains.
- A fully on-chain trustless price oracle that rewards beacon operators for calling the contract to store price history for multiple LP's, which benefits our current users by significantly reducing gas fees, and provides a verifiable on-chain price for other DAPP DeFi developers.
- Integration of custom TradingView signals to capture institutional/Quant funds that have developed their own internal strategies.