Sprawl

A permissionless and censorship resistant peer-to-peer network for decentralised exchange

Patricio Palladino, Franco Zeoli, Teemu Paivinen
September 25th, 2018

DRAFT 0.5

Abstract

We describe a peer-to-peer exchange network which maintains a decentralised order book and executes trades on the Ethereum blockchain. The network is intended to facilitate a fair and open market for crypto-assets without the monopolistic rent-seeking tendencies of centrally managed exchanges. The network is permissionless, allowing anyone in the world to participate, censorship resistant, in that no single party can influence what is traded, and scalable, as it is not constrained by a consensus mechanism until trade execution.

Table of Contents

Abstract **1**

Table of Contents **2**

Introduction **3**

Existing Work **4**

Sprawl **5**

- Nodes 5
- Trading Client 6
- Scalability 6
- Permissionless-ness, Censorship Resistance and Custody 7
- Order Matching 10
- Fees and Incentives 10
- Summary 12

References **13**

Introduction

The concept of decentralised, digital value was first realized with the introduction of Bitcoin in 2009. By combining public key cryptography with a consensus algorithm, Bitcoin was able to solve the double spend problem [1], while remaining permissionless. This construction was the first system for keeping track of value, which allowed anyone to participate in the consensus process of maintaining and deciding on the state of the ledger without requiring a central authority.

Inspired by Bitcoin, a number of similar mechanisms for transferring value emerged soon after. Decentralised value storage and transfer, could be seen as the core mechanisms provided by this first wave of cryptocurrency innovation.

Then, in 2015, Ethereum introduced its blockchain with a Turing-complete programming language, allowing developers to create “smart contracts” with arbitrary rules for ownership, transaction formats and state transition functions [2]. As developers created more complex constructions with smart contracts, the resulting product-like services came to be known as decentralised applications or dApps.

While dApps are more marketable, the core innovation, which spawned this second wave of innovation, was the smart contract; the ability for economic participants to enter into simple enforceable contracts without the need for a trusted third party. This conceptually simple idea spawned thousands of new projects, testing the edges of what could be achieved with the current infrastructure.

One of the directions of this exploration is the concept of decentralised exchange [3]. The intention of decentralised exchanges is to create global, decentralised markets for exchanging digital tokens, facilitating price discovery and more complex forms of commerce.

In many ways, the purpose of trustless systems is to allow a completely decentralised economy to emerge. An economy of any kind first needs value, a token or asset which can be used as a unit of account. Second, it needs contracts, the ability to enter into agreements and obligations, facilitating simple commerce. Third, it needs functional markets, which unlock more complex forms of commerce, more efficient capital allocation and the free flow of value through the economy.

While a lot of effort has been put into creating decentralised markets, they have yet to be realized in a functional form, even on a single blockchain platform like Ethereum. This is primarily due to the scalability limitations of blockchain-based systems, caused by the need for a consensus mechanism. These limitations may be overcome with time, but there might also be alternative technologies which could be used to deliver scalable, decentralised markets considerably sooner.

In this paper we propose Sprawl, a peer-to-peer network which facilitates scalable, decentralised exchange of Ethereum-based assets, while remaining permissionless and censorship resistant.

Existing Work

Decentralised exchanges which use atomic swap smart contracts, and are primarily aimed at ERC20 and other Ethereum token standards, have been implemented in a variety of ways. These implementations broadly fall into two groups, depending on how the order book and order matching has been implemented. All decentralised exchanges use or provide on-chain trade execution, while orderbooks are implemented either on-chain, using smart contracts, or off-chain in a centralized fashion.

On-chain order book implementations, such as OasisDEX [4], are constrained by Ethereum's scalability, as all interactions with the orderbook incur a gas fee and are relatively slow due to speed of the consensus mechanism. Due to their limitations, on-chain order books generally only make sense for use-cases where decentralisation is a considerably higher priority than throughput or user experience. However, on-chain implementations are legitimately decentralised and thus censorship resistant, in that no central authority can influence which assets are traded or interfere with trading activity in any way.

Off-chain order books scale considerably better, being functionally no different to those of centralised exchanges. Interactions with the order book are swift and free. As interactions are not limited by consensus requirements, exchanges utilizing this architecture can provide throughput similar to that of centralised exchanges. Trade execution remains a bottleneck, but the throughput requirements for execution alone are considerably smaller. Furthermore, in most cases the slight delay in settlement occurs once a trade is guaranteed, leading to minimal impact on user experience. These exchanges, however, sacrifice decentralisation almost entirely, as they maintain centralised control of the order book and order matching [5][6][7]. Implementations such as this cannot feasibly provide censorship resistance and thus are susceptible to all manner of interference. A fact demonstrated when IDEX blocked trading from New York on October 24th, 2018 [8].

Neither of these implementations achieves the properties necessary to facilitate scalable, decentralised markets in the near term. On-chain order books simply cannot provide the throughput or user experience necessary, while off-chain order books sacrifice the very property decentralised exchanges promote themselves as having.

As far as we are aware, a decentralised order book has only been implemented by Bisq [9], previously known as Bitsquare. In Bisq a decentralised order book protocol is used to facilitate trades between cryptocurrency and fiat. While conceptually in a promising direction, Bisq's implementation suffers from the need to facilitate fiat transfers and a legacy architecture. Bisq

was originally built for exchanging bitcoin and fiat, which meant that it was implemented without knowledge of the current smart contract paradigm. This leads to a user experience which cannot be said to resemble centralised exchanges, or even the decentralised exchanges we have today.

Sprawl

Sprawl is a decentralised exchange network. An open, non-custodial, anonymous, permissionless and censorship-resistant public utility for trading crypto-assets. It belongs to no one and benefits everyone. Trading fees are used for the benefit of the network as a whole, by directly incentivizing liquidity and ecosystem service providers. In addition to a default open-source trading client, anyone can build new trading software and user interfaces for the network to provide tools aimed at different kinds of investors.

The network consists of a traditional peer-to-peer network where each node holds a full replica of the order book. Nodes communicate to relay the buy and sell orders that compose the order book, in a similar fashion to how peer-to-peer file-sharing networks work. When a node matches an order the trade is executed trustlessly using a robust on-chain trade execution protocol. In its initial configuration, Sprawl will support the trading of any Ethereum-based asset and use the 0x protocol for execution.

The network runs in parallel to the Ethereum blockchain to solve the scaling impossibility of having an order book on-chain, while still being able to leverage the security and smart contract functionality a blockchain provides. There are no compromises on decentralisation.

Communication between nodes is fully encrypted and the data transmission is done on a best-effort basis, making the network eventually consistent. There is no consensus mechanism, making scaling a possibility without the need for layer-2 solutions.

Nodes

The node software will run both as a daemon for infrastructure use-cases and alongside the default open-source trading client for retail users, making every direct user a node in the network. Using a similar strategy the BitTorrent network reached a peak of twenty million nodes running simultaneously without issues [10]. A modified Ethereum node will be bundled together with the Sprawl node to watch the Ethereum mempool and react quickly to upcoming events.

The nodes primarily perform four services for the network:

1. Validating every order they receive and dropping invalid ones.
2. Relaying valid orders they receive to other nodes.
3. Broadcasting new orders to the network.

4. Detecting malicious nodes and dropping them.

To leverage the existing software used for trading on 0x, the Sprawl node will expose a local API that follows the 0x Standard Relayer API specification [11]. This will allow any algorithmic traders operating on 0x relayers to trade on Sprawl with no additional integration effort or cost.

Trading Client

The default Sprawl trading client is an open-source desktop interface for interacting with the network. It consists of a Sprawl node, a modified Ethereum node, and the trading UI. Its purpose is to make it easy for retail users to begin trading on Sprawl by providing a clear and familiar exchange experience. The client is designed to be extendable, so that developers can easily build new features, as well as forkable, allowing new clients to be built quickly to serve different demographics.

Users will download the Sprawl software, run it and access the trading client through their browser of choice. They will then choose the assets they are interested in trading and the Sprawl node will synchronize with other nodes which have the desired order books. This initial download of the order books should only take a few seconds and once it is complete, the user will be able to trade directly from MetaMask or Ledger wallets, as with any non-custodial exchange.

We are evaluating building on WebRTC, which coupled with services like Infura and The Graph would enable a Sprawl trading client to run completely in the browser. However, these technologies are currently missing critical functionality for this use-case. Until then, a pure in-browser end-user experience will require a node operator to offer to relay orders to the network as a service.

Scalability

Given that we've seen several peer-to-peer networks reach significant scale when there isn't a consensus mechanism getting in the way and because order data is comparatively small in size, we believe Sprawl won't face substantial scaling issues until very significant scale is reached. We don't know where this milestone is exactly, but expect it to be far enough where scalability doesn't need to be a focus of early development.

Until then we will primarily focus on optimizing latency and bandwidth requirements for nodes, in order to minimize the requirements of participating in Sprawl, which will hopefully lead to an increase in decentralisation. To do this, we are exploring sharding the network by trading pair, meaning that nodes will subscribe to the pairs they are interested in and connect directly with peers who are holding the desired order books. This could lead to more efficient routing, as well as greater flexibility in storage requirements.

There are some security implications to this approach. If a malicious actor is able to get a list of all the nodes participating in a given pair, they may have an opportunity to perform a DDoS attack on the participants and disrupt trading activity for a specific pair. We are exploring different approaches to protecting nodes from this kind of attack, including fuzzy routing algorithms.

Permissionless-ness, Censorship Resistance and Custody

Sprawl is exclusively a peer-to-peer order book, with an organic incentive structure for matching trades and executing them on-chain. It is permissionless by design, in that anyone can participate and fill any role in the network according to their needs and interests.

There is no registration process, no accounts, so users are anonymous by default. There is also no listing process for assets and the only hard constraint for which assets can be traded, is whether viable trade execution protocols exist. These properties will persist because in a peer-to-peer network all nodes would need to be coerced for these properties to change. At scale, the cost and complexity involved in coercing every node operator to make a change is untenable even for nation-state actors.

Sprawl is also non-custodial, in that the assets being exchanged move directly from wallet to wallet with no middleman in between. This is done using the 0x protocol, which grants that layer the same security and censorship resistance properties as Ethereum.

Order Matching

The Front-running Problem

There is a known problem in decentralised exchanges, normally referred to as the “front-running problem” [12]. It’s essentially a race condition that impacts UX in a negative way. When someone takes an order, someone else could be attempting to take the same order and until one of the two execution transactions is mined into the next Ethereum block, neither of the traders know if the trade was successful, and only one will be. Several factors come into play to determine which of the colliding trades succeeds.

The main issue is not a trade failing, as that also happens in centralized exchanges, but the delay between the user attempting a trade and receiving notice of the trade failing. In addition to the negative effect on user experience, this mechanism is susceptible to front-running attacks where a malicious actor, knowing that someone is taking a trade, could forcefully take it first by sending an Ethereum transaction with a larger gas fee.

Using a 0x matching strategy [13] and some specific communication functionality in the protocol we can prevent this race condition. In 0x v2 [14], there are three fields in each order that define the different parties involved in a trade: the maker, the taker and the sender. The first two are self-explanatory. The sender is a third actor who is authorized to act on behalf of the maker when it comes to actions that affect the order, such as setting the taker and executing the order.

The mechanism to prevent front-running works as follows:

1. A node posts a make order into the network, using the sender field to only allow itself to set the taker field. This prevents others from executing the order without the maker node's intervention.
2. Trader #1 sees the order in the order book, and sends a message into the network communicating that they want to take it, alongside a signature that is ready for execution.
3. The take message goes through the network, making the nodes remove the taken order from the order book and watch the mempool for a transaction that executes the trade.
4. The take message is received by the maker node, who sets the taker field and sends a transaction to the blockchain to execute the trade.
5. The network sees the transaction executing the trade in the mempool and stops watching for its execution.

This is the basic case. If trader #2 also wants to take the order at the same time, sending a conflicting take message to the network, the maker node, having already received the take message from trader #1, notifies trader #2 that the order has already been taken. This should take a few seconds, rather than the one block confirmation time it would take without a mechanism to prevent trades colliding.

The race condition feedback delay is improved significantly through this mechanism where each node acts as a Trade Execution Coordinator (TEC) [15] for its own make orders, achieving essentially a decentralised TEC across Sprawl nodes. This mechanism also causes some positive side effects. Orders can only be matched and executed when the node responsible for publishing the order is online. This has the positive consequence of protecting nodes experiencing connection problems from erroneous trades being executed while they are offline. Another positive side effect stems from the fact that maker nodes pay for the gas fees on every trade. This means that taking doesn't require the participating party to hold ETH, allowing for a significant improvement in user experience.

Cancelling Orders

To perform a cancellation at the 0x protocol level an Ethereum transaction is needed. In the context of centralized relayers who are using the matching strategy, one can simply take an order off the order book and this achieves the desired effect of cancellation with immediacy. In

an open order book model [16], simply removing the order from the order book does not protect the trader from trading bots storing the order and executing it later.

In the case of Sprawl, given the decentralised context, cancelling is even trickier as there is no simple control mechanism by which an order can be removed from simultaneously from every node's order book. Without attempting to improve on the available mechanisms, the two basic options are the Ethereum transaction or broadcasting a message into the network communicating that an order is being cancelled and should be removed from the order book.

The problems with an on-chain transaction are the unacceptable delay in cancellation resulting from block times and excessive friction in usability. Simply broadcasting a cancellation message, on the other hand, doesn't guarantee that the order is removed from the order book by other nodes, rendering this method similarly unrealistic.

Instead, the mechanism used to address front-running in Sprawl can also be used for order cancellation to achieve a fast and low friction user experience. To cancel an order a node can immediately start ignoring take messages from other nodes. When there is no trade execution transaction in the mempool immediately following a take message, the network can remove the order from the order book, assuming that the order has been cancelled. This achieves immediate off-chain order cancellation, guaranteeing an order will not execute as soon as the canceling party makes the decision and the removal of the order from the order book shortly thereafter.

The cancelling node should still send a cancellation message into the network so that the order is removed from the order book even if no one is trying to take it, but the network should not be reliant on this.

Order Spoofing

One downside to this matching model is that without extra preventive measures it facilitates order spoofing, a known manipulation tactic that is common on centralized exchanges.
A node could send a large order and refuse to match it with prospective takers, misleading the market.

We are exploring different methods to detect this behaviour. As a starting point nodes can watch for the *cancel order* message that should briefly follow an ignored *take order* message, and based on repeated occurrences of this sequence happening or the complete lack of a *cancel order* message the node can be assumed to be dishonest.

While order spoofing is a not to be taken lightly, it continues to be an issue in most centralized exchanges as well. Achieving the same levels of market integrity as the current state of the art while completely decentralising the platform is in our view an acceptable initial outcome.

Order Matching Requirements

A requirement that arises from the order matching model employed in Sprawl, is that nodes publishing make orders must remain online at all times as they are responsible for matching and executing said order. For nodes providing liquidity or engaging in algorithmic trading, this is unlikely to be an issue as they generally need to remain online anyway in order to respond to changes in the market.

However, this may be something retail users don't want to deal with. As explained later on in the incentives section, this creates a valuable service that node operators can provide to Sprawl users for a profit.

Market Orders

In the recently launched 0x protocol v2, it is possible to take several orders in one atomic operation. For the first time, this allows the execution of real market orders on 0x, as long as the open order book strategy is used. Due to the front-running problem described previously, like DDEX, Ethfinex and others, Sprawl uses the matching strategy. This prevents Sprawl from using the market order functionality in the 0x protocol, as the maker nodes need to be involved in the process of their trades being executed. The way market orders work in Sprawl is through the taking of several orders in a sequential fashion until the entire order is filled. While wasteful in terms of gas, this doesn't mean an increase in cost for the taker, as that cost is assumed by the node which published the make order.

Fees and Incentives

Centralized exchanges exhibit monopolistic and rent-seeking tendencies, driven primarily by fee-based business models combined with liquidity network effects. In Sprawl there is no exchange operator and as such the network itself is not rent-seeking.

Sprawl recognizes three actors involved in the fee structure: makers, takers and nodes. The most important dynamic to understand in regards to how fees work is that fees are always paid to the node operators, even if the node operator happens to also be the maker or taker.

Every order on Sprawl is validated to strictly follow the fee protocol rules for acceptance into the network. The rules enforce a 0.1% fee on the taker on every order, without exception. The node operator who published the make order collects the payment.

In addition to this fee, node operators are allowed to charge their users a fee on top. This means that a node that sources liquidity can charge the maker, and a node that sources liquidity consumption can charge the taker.

For example, a node is providing a relayer service and charges an additional 0.1% fee. In this case, there are two beneficiaries: the node operator who originally published the order receives the 0.1% protocol fee and the node operator who sourced the taker receives the additional 0.1% taker fee. This is the only case where there is fee revenue going to a node that is not the publisher of an order.

These last two rules enable a new kind of relayer business model within the protocol, as fees can be charged for providing a user interface with additional services, order matching and operating a node.

Considering that both makers and takers can run their own nodes without additional effort by simply using the open source trading client, there are five possible scenarios with differing fee revenue outcomes:

- Scenario A
  Maker is using his own node and a taker using his own node takes his order.

- Scenario B
  Maker is using his own node and a taker using a third-party node takes his order.

- Scenario C
  Maker is using a third-party node and a taker using his own node takes his order.

- Scenario D
  Maker is using a third-party node #1 and a taker using a third-party node #1 takes his order.

- Scenario E
  Maker is using a third-party node #1 and a taker using a third-party node #2 takes his order.

To get a better sense of what this looks like, let us say that a 0x relayer operating on top of Sprawl that plays the part of the above mentioned third-party node charges takers a fee TF and makers a fee MF. Being up to the relayer, either or both of these could be zero. The different scenarios play out like this:

- Scenario A
  Maker is using his own node and a taker using his own node takes his order.
  Taker pays 0.1%. Maker collects.

- Scenario B
  Maker is using his own node and a taker using a third-party node takes his order.
  Taker pays 0.1% + TF. Maker collects 0.1% and relayer collects TF.

- Scenario C
  Maker is using a third-party node and a taker using his own node takes his order.
  Taker pays 0.1% and maker pays MF. Relayer collects both.

- Scenario D
  Maker is using a third-party node #1 and a taker using a third-party node #1 takes his order.
  Taker pays 0.1% + TF and maker pays MF. Relayer collects both.

- Scenario E
  Maker is using a third-party node #1 and a taker using a third-party node #2 takes his order.
  Taker pays 0.1% + TF and maker pays MF. Relayer #1 collects MF + 0.1% and relayer #2 collects TF.

Keep in mind that both MF and TF are optional, and in many cases will be zero.

The main objectives behind this structure are:

1) To incentivise liquidity providers to make markets.
2) To provide revenue generation opportunities to node operators.
3) To incentivise, but not require, all participants to run nodes.

As there can be no network effects from liquidity, since everyone shares a single liquidity pool, relayers must provide additional value to justify additional fees. Not needing to sacrifice liquidity means the cost of switching relayer is close to zero and users always have the option of running a node themselves, where they only pay the standard taker fee enforced by the protocol. This creates competitive dynamics where relayers must be providing significant additional value to justify charging additional fees or risk losing their users to a cheaper relayer.

Over time we expect these dynamics to lead to higher quality services and lower costs for end-users, while allowing liquidity providers and node operators to maintain sustainable revenue models.

Summary

Sprawl was conceived to facilitate the emergence of truly global decentralised markets, by providing permissionless, censorship resistant and scalable exchange for crypto-assets.

As a short-term consequence, Sprawl allows for improved economics for all parties involved, achieved by removing the middle-man, and greater accessibility to all Ethereum-based assets, by removing barriers to entry.

As a long-term consequence of scalable, decentralised markets, considerably more complex forms of commerce, ownership and value creation become possible. A genuinely decentralised exchange facilitates pricing efficiency, enables the efficient flow of capital through the ecosystem and consolidates liquidity without creating network effects for a single operator.

Furthermore, Sprawl is open-ended by design, allowing for the creation of many kinds of valuable third party services, and can be extended to support new asset standards or blockchain platforms. We believe it will provide a stable, yet flexible, foundation for decentralised financial markets in the trustless economy being built.

References

[1] Bitcoin Whitepaper https://bitcoin.org/bitcoin.pdf
[2] Ethereum Whitepaper https://github.com/ethereum/wiki/wiki/White-Paper
[3] 0x Whitepaper https://0xproject.com/pdfs/0x_white_paper.pdf
[4] OasisDEX https://github.com/OasisDEX/oasis/wiki
[5] DDEX.io https://ddex.io/
[6] Radar Relay https://radarrelay.com/
[7] EthFinex https://www.ethfinex.com/
[8] Idex Blacklists New York Residents Setting A Worrying Precedent
https://news.bitcoin.com/idex-blacklists-new-york-residents-setting-a-worrying-precedent/
[9] Bisq https://docs.bisq.network/
[10] L. Wang and J. Kangasharju. Measuring large-scale distributed systems: case of bittorrent mainline dht. In Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on, pages 1–10. IEEE, 2013. http://128.214.112.31/u/lxwang/publications/P2P2013_13.pdf
[11] 0x Standard Relayer API https://github.com/0xProject/standard-relayer-api
[12] Front-running, Griefing and the Perils of Virtual Settlement (Part 1)
https://blog.0xproject.com/front-running-griefing-and-the-perils-of-virtual-settlement-part-1-8554ab283e97
[13] 0x Matching Strategy https://0xproject.com/wiki#Matching
[14] 0x v2 Order Format
https://github.com/0xProject/0x-protocol-specification/blob/master/v2/v2-specification.md#order-message-format
[15] Front-running, Griefing and the Perils of Virtual Settlement (Part 2)
https://blog.0xproject.com/front-running-griefing-and-the-perils-of-virtual-settlement-part-2-921b00109e21
[16] 0x Open Order book Model https://0xproject.com/wiki#Open-Orderbook