Stocksd.app: on-chain oracle-based stock trading

contact@stocksd.app
stocksd.app

July 9, 2020

Abstract

This paper describes a blockchain protocol that enables decentralized (P2P) stock trading on Ethereum (EVM) platform. The protocol is specifically tailored towards stock trading and is largely unsuitable for most of the other financial instruments.

The core of the protocol relies on oracle requests and the exchange contract, therefore allowing parties to match and trade trustlessly and without a third-party arbiter. This paper addresses technical architecture as well as economic and legal framework for stocksd.app implementation.

Contents

1 Introduction

The rapid advent of blockchain oracle technology has enabled efficient on-chain price tracking of any asset for which it is possible to retrieve a reliable price feed [2]. This opened possibilities of betting and trading in a way that doesn't involve a trusted third party. However, despite technological maturity, there are economic (behavioral) and legal challenges specific to each asset class.

In this paper, we outline a protocol that addresses stock trading specifically. There are a number of striking differences between stock and cryptocurrency market: limited trading hours and holidays, brokerage authorizations, absence of fractional trading by default, stock class difference, trading halts and others. In addition to that, stock trading is subject to both supranational and local law (e.g. AML/KYC procedures, taxation regulation and PIL) [3] [1]. Current whitepaper strives to address these issues comprehensively in stockd.app protocol description but remains subject to further continuous amendments.

2 Stocksd.app protocol overview

Stocksd.app central abstractions are that of seller (or dSeller, referred interchangeably) and buyer roles. Both roles are not mutually exclusive and can be shared by a single Ethereum address.

The central element of the protocol is the `Exchange` smart contract which governs open offers and trades and performs oracle requests. Functionally it structures trade flow like a DAG (directed acyclic graph), where each trade follows a typical lifecycle.

1. To initialize an offer, a seller has to provide a deposit of any amount of stablecoin (DAI in the current protocol implementation) that will serve as a collateral against asset appreciation. In other words, if the price of a stock goes up this would be the deposit that a buyer will be paid from. No price request is made upon offer initialization.
2. After an offer is successfully opened, a potential buyer can accept it via fulfilling the offer at the current market price using price oracle request. Buyer deposit (denominated in DAI as well) covers the total value of stock purchased plus seller commission. This deposit will serve as a collateral against full asset depreciation.
3. After a price oracle request is fulfilled, deposits are locked and the seller is expected to procure advertised stock in a timely manner. Delays in procuring advertised stock will proportionally increase volatility risk for the seller.
4. At this stage (right after an oracle request is fulfilled) the stock buyer is free to close a position at any time but subject to the restrictions described in `Volatility and front-running protections` subsection. The seller incurs volatility risks identical to those of procuring advertised stock when an offer is accepted.
5. Closing a position triggers a second oracle request which determines the final value of position. If a trade turns out to be profitable, the stock buyer receives profit from seller deposit plus initial deposit, commission; deposit leftovers are returned to the seller. If a trade turns out to be unprofitable, the stock buyer receives initial deposit at the current price; commission plus position loss coverage are returned to the seller from buyer collateral.

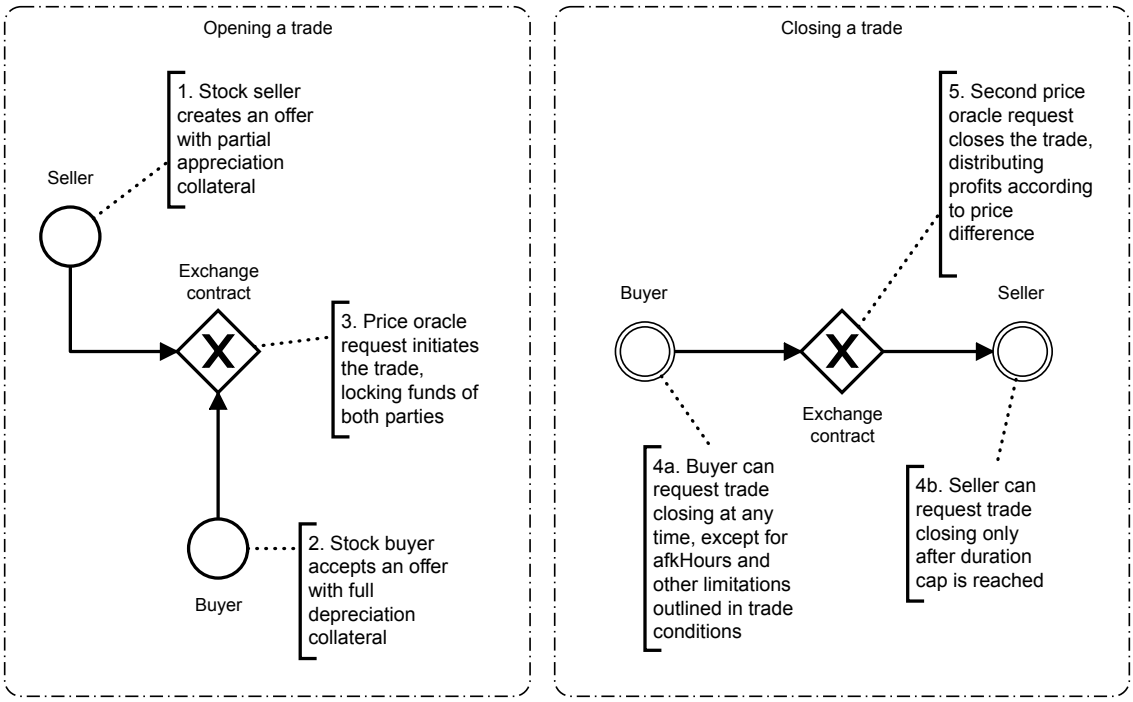

Figure 1: Trade lifecycle diagram

3 Exchange contract

Current protocol implementation is written in Solidity, therefore data structures required by protocol are described in terms of Solidity types [4].

Table 1: On-chain trade descriptor fields (Solidity). `!` denotes a required field

Field	Meaning	Variable type
`ticker!`	Stock ticker for oracle	`bytes32`
`initPrice!`	Oracle-supplied initial accepted price	`uint`
`buyer!`	Buyer authorization	`address`
`seller!`	Seller authorization	`address`
`amountStock!`	Stock amount purchased	`uint`
`fundsSeller!`	Stablecoin appreciation collateral amount	`uint`
`afkHours`	Timespan when accepting or closing is disabled	`uint[]`
`durationCap`	UNIX timestamp limit of trade duration	`uint`

3.0.1 Trade representation

Along with supporting/metadata fields, stocksd.app Exchange contract should be able to persist some core information about any given trade.
Ticker field is used in open-trade and close-trade oracle requests to identify and query stock price.
`initPrice!` field is populated by open-trade oracle request and used in profit/commission estimation formula when a trade is closed.
`fundSeller!` field is a seller-controlled field that keeps track of appreciation deposit. There are no constraints on this field other than requirement of a positive integer.

3.0.2 Volatility and front-running protections

For any given trade, a seller is able to set `afkHours` field that is checked whenever a buyer tries to accept or close a trade. This check shields seller from volatility risk during a gap between action performed on-chain and actual stock market operations.
`durationCap` is another risk-minimization feature intended to protect seller interests. After `durationCap` threshold is passed, a seller is able to close the trade on buyer's behalf.
Front-running opportunity opens up when accepting or closing a trade. To protect parties from front-running impact, on-chain action sanity checks (response boundaries) are implemented. A party launching an oracle request is able to select response boundaries seamlessly - oracle response should fail if data boundary (e.g. heavy price slippage) is exceeded, thus making repeated front-running attempts uneconomical.
To mitigate general volatility, all protocol deposits are processed in stablecoin (DAI [5]).

3.0.3 Oracles

Stocksd.app relies on oracles to govern trade outcome. In the current implementation two requests are made to Chainlink [2] stock price oracle contract and an action is performed (finalized) only if an oracle callback responds with a non-error code. Oracle response fault-tolerance is based on sufficiently long timeouts. When an oracle request reaches its timeout, the action can be re-requested.

```
*this section is incomplete*
```

4 Legal framework [1]

Stocksd.app aims to comply with all supranational and local regulations as a P2P Dapp. Accordingly, parties to a trade manage their own compliance with any laws applicable to them in their respective jurisdictions.
Stocksd.app protocol optionally provides adequate tools to request KYC/AML data from a party to a trade, but it will not universally oblige any party to go through it. Due to it's P2P nature, stocksd.app trade requirements are priced in for each trade and one can assume that a seller offering lax or privacy-friendly retail trades will be able to attract larger amount of buyers.
Functionally, stocksd.app protocol is aiming to be as on-chain and decentralized as possible. Thus an implementation of the protocol is jurisdiction-agnostic, privacy-first and censorship-resistant.

```
*this section is incomplete*
```

5 Summary

1. Stocksd.app is a decentralized application (Dapp or dApp) that enables decentralized (P2P) oracle-based stock trading on the Ethereum network.
2. The ecosystem consists of the exchange contract (updated on a rolling basis), oracle nodes and an oracle contract supplying individual stock data.
3. There are a number of features that stocksd.app protocol implements to protect the interests of the seller who incurs volatility risks that come with real-life stock procurement (duration caps, afkHours parameters, stablecoin usage).

6 Appendix

References

[1] James Steven Rogers. Conflict of Laws for Transactions in Securities Held through Intermediaries. https://scholarship.law.cornell.edu/cgi/viewcontent.cgi?article=1670&context=cilj, 2006. Accessed: 2020-03-17.

[2] Steve Ellis, Ari Juels, Sergey Nazarov. https://link.smartcontract.com/whitepaper, 2017. [Online; accessed 8-May-2020].

[3] The Regulation of Trading Markets: A Survey and Evaluation. https://repository.law.umich.edu/cgi/viewcontent.cgi?article=1121&context=book_chapters, 2018. Accessed: 2020-04-11.

[4] Solidity documentation. https://solidity.readthedocs.io/en/latest/. Accessed: 2020-04-10.

[5] MakerDAO Whitepaper. https://makerdao.com/whitepaper/WhitePaper-TheMakerProtocol_MakerDAO\OT1\textquoterightsMulti-CollateralDai(MCD)System-FINAL-021720.pdf. Accessed: 2020-07-20.