TBLC: An open source decentralized exchange protocol on top of the Binance blockchain

Zeud J. Honi , Valtazar Zutic
tblcexchange.com

March 19, 2021

Abstract

The TBLC exchange platform is an upcoming decentralized exchange platform exclusively built to exchange the crypto assets deployed on the Binance smart chain. Initially the project will focus around distribution of TBLC tokens among the community. The TBLC tokens can be availed in exchange of the BNB, the native asset of Binance blockchain ecosystem. Binance smart chain is a complementary system to the existing Binance Chain. To facilitate inter-blockchain transferability, the Binance smart chain uses dual chain architecture. This makes the platform a powerful platform to facilitate rapid decentralized trading. The platform is being seen as an option to the current scalability and financial challenges put forth by ethereum. While ethereum is a major leap towards "Web 3.0", increasingly hefty transaction fees has kept the blockchain ecosystem to reach the hands of communities with lower per capita income. BNB has recently launched its own smart contract platform as an option to the ethereum ecosystem. We describe a protocol that facilitates low friction peer-to-peer exchange of BEP20 tokens on top of the Binance blockchain. The protocol is intended to serve as an open standard and common building block, driving interoperability among decentralized applications (dApps) that can be deployed on top of Binance blockchain. Trades are executed by a system of BEP20 open standard smart contracts that are publicly accessible, free to use and that any dApp can hook into. DApps built on top of the Binance Smart Chain can access public liquidity pools or create their own liquidity pool and charge transaction fees on the resulting volume. Decentralized governance is used to continuously and securely integrate updates into the base protocol without disrupting dApps or end users.

Contents

1 Introduction

Blockchains have been revolutionary by allowing anyone to own and transfer assets across an open financial network without the need for a trusted third party. Now that the Binance blockchain ecosystem has taken a major leap towards the dapp ecosystem, a number of applications are bound to be developed on Binance chain. With the advent of decentralized applications on Binance blockchain, the Binance decentralized ecosystem is bound to grow at a rapid speed now. As more and more asset protocols will enter into the BNB ecosystem, there was a need so that these assets can be exchanged without the need for a trusted third party.

Decentralized exchange is an important progression towards the development of the BNB blockchain: First, decentralized exchanges can provide stronger assurance in terms of security to project owners as well the end users since there is no longer a central party which can be hacked, run away with customer funds or be subjected to government regulations. Second, Having a decentralized application will create this trustless ecosystem more trusted for the end users.

In the BNB blockchain's journey towards decentralized computing and supporting smart contracts, a possibility of lack of best practices may leave end users exposed to numerous smart contracts of varying quality and security, with unique configuration processes and learning curves. This approach may impose unnecessary costs on the network by fragmenting end users according to the particular dApp each user happens to be using, destroying valuable network effects around liquidity.

TBLC is an open protocol for decentralized exchange on the BNB blockchain. It is intended to serve as a basic building block that may be combined with other protocols to drive increasingly sophisticated dApps on BNB. TBLC will use a publicly accessible smart contracts infrastructure of different decentralized applications on BNB. In the long run, open technical standards will tend to win the closed ones, and as more and more assets will be deployed on the BNB blockchain, we will see more dApps that require the use of these assets. As a result, A decentralized token exchange is critical to supporting this open economy.

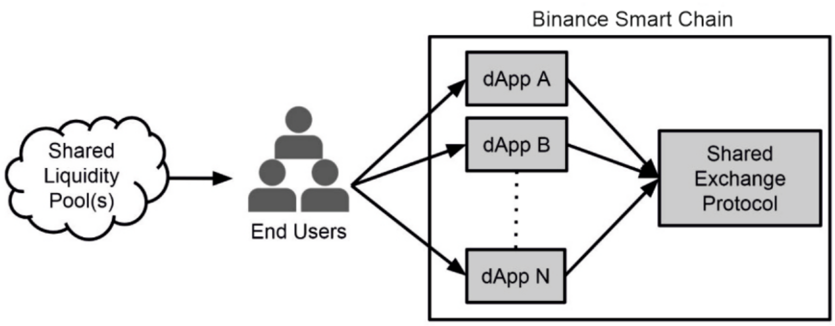

Figure 1: Open protocols should be application-agnostic. Decoupling the protocol layer from the application layer provides mutual benefits for dApp developers and end users alike.

2 Existing Work

Decentralized exchanges implemented with smart contracts based on different blockchains have failed to generate significant volume due to inefficiencies in their design that impose high friction costs on market makers. In particular, these implementations place their order books[1] on the blockchain, requiring market makers to spend gas each time they post, modify or cancel an order. While the cost of a single transaction is small, frequently modifying orders in response to evolving market conditions is prohibitively expensive. In addition to imposing high costs on market makers, maintaining an on-chain order book results in transactions that consume network bandwidth and bloat the blockchain without necessarily resulting in value transfer.

Side chains are a proposed scalibility solution for the existing blockchains and reducing costs for a variety of applications - including exchange [9] - by moving transactions off of the blockchain [10,11]. Participants in a state channel pass cryptographically signed messages back and forth, accumulating intermediate state changes without publishing them to the canonical chain until the channel is closed. State channels are ideal for “bar tab” applications where numerous intermediate state changes may be accumulated off-chain before being settled by a single on-chain transaction (i.e. day trading, poker, turn-based games). If one of the channel participants leaves the channel or attempts to cheat, there is a challenge period during which the other participant may publish the most recent message they received from the offender. It follows that channel participants must always be online to challenge a dishonest counterparty and the participants are therefore vulnerable to DDOS attacks. While state channels drastically reduce the number of on-chain transactions for specific use cases, the numerous on-chain transactions and security deposit required to open and safely close a state channel make them inefficient for one-time transactions.

A hybrid implementation, which we refer to as “off-chain order relay with on-chain settlement,” combines the efficiency of state channels with the near instant settlement of on-chain order books. In this approach, cryptographically signed orders are broadcast off of the blockchain; an interested counterparty may inject one or more of these orders into a smart contract to execute trades trustlessly, directly on the blockchain. Friction costs are minimized for market makers because they can signal intent off-chain and transactions only occur when value is being transferred. We extend this approach by allowing anyone to act as the exchange and by making the protocol application-agnostic.

[1] An order book is used to publicly record the interest of buyers and sellers in a particular financial instrument. Each entry includes a reference to the interested party, the number of shares and the price that the buyer or seller are bidding/asking for the particular security.

3 Specification

Figure 2 presents the general sequence of steps used for off-chain order relay and on-chain settlement. For now we ignore a few mechanisms that will become important later.

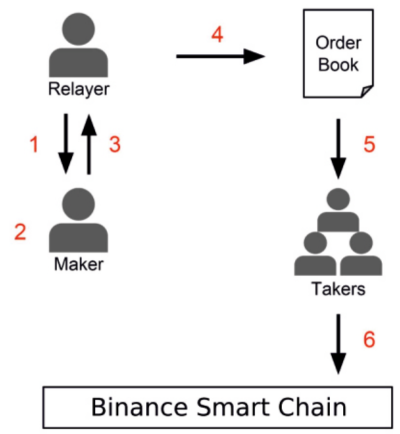

Figure 2: Off-chain order relay, on-chain settlement diagram. Gray rectangles and circles represent BNB smart contracts and accounts, respectively. Arrows pointing to BNB smart contracts represent function calls; arrows are directed from the caller to the callee. Smart contracts can call functions within other smart contracts. Arrows external to the BNB blockchain represent information flow.

1. Maker approves the decentralized exchange (TBLC) contract to access their balance of Token A[2].
2. Maker creates an order to exchange Token A for Token B, specifying a desired exchange rate, expiration time (beyond which the order cannot be filled), and signs the order with their private key.
3. Maker broadcasts the order over any arbitrary communication medium.
4. Taker intercepts the order and decides that they would like to fill it.
5. Taker approves the TBLC contract to access their balance of Token B.
6. Taker submits the maker's signed order to the TBLC contract.
7. The TBLC contract authenticates maker's signature, verifies that the order has not expired, verifies that the order has not already been filled, then transfers tokens between the two parties at the specified exchange rate.

[2]See BEP20 Token in Appendix. It is possible to provide approval once and execute an unlimited number of trades thereafter. Alternatively, one can provide approval prior to - and limited to the value of - each individual trade.

3.1 Message Format

Each order is a data packet containing order parameters and an associated signature. Order parameters are concatenated and hashed to 32 bytes via the Keccak SHA3 function. The order originator signs the order hash with their private key to produce an ECDSA signature.

Point-to-point orders allow two parties to directly exchange tokens between each other using just about any communication medium they prefer to relay messages. The packet of data that makes up the order is a few hundred bytes of hex that may be sent through email, a Facebook message, whisper or any similar service. The order can only be filled by the specified `taker` address, rendering the order useless for eavesdroppers or outside parties.

Table 1: Message format for point-to-point orders.

`version`	`address`	Address of the Exchange smart contract. This address will change each time the protocol is updated.
`maker`	`address`	Address originating the order.
`taker`	`address`	Address permitted to fill the order.
`tokenA`	`address`	Address of a BEP20 Token contract.
`tokenB`	`address`	Address of a BEP20 Token contract.
`valueA`	`uint256`	Total units of tokenA offered by maker.
`valueB`	`uint256`	Total units of tokenB requested by maker.
`expiration`	`uint256`	Time at which the order expires (seconds since unix epoch).
`v`	`uint8`	ECDSA signature of the above arguments.
`r`	`bytes32`	
`s`	`bytes32`	

For liquid markets to emerge, there must be public locations where buyers and sellers may post orders that are subsequently aggregated into order books i.e. exchanges. Building and operating an exchange is costly and the protocol we have described so far does not provide an incentive for someone to take on such an expense. Broadcast orders solve this issue by allowing anyone to act as an exchange, maintain an order book (public or private) and charge transaction fees on all resulting liquidity. We refer to entities that host and maintain an order book as Relayers rather than exchanges. Where an exchange must build and operate proprietary infrastructure, execute trades and handle user funds, Relayers merely facilitate signalling between market participants by hosting and propagating an order book that consists of generic messages. Relayers do not execute trades on behalf of market participants as this would require market participants to trust the Relayer. Instead, Takers execute their own trades.

The message format for broadcast orders includes two changes to the point-to-point message format to facilitate public exchange and incentivize Relayers. First, broadcast orders do not specify a `taker` address, allowing a broadcast order to be filled by anyone that happens to intercept it. Second, broadcast orders include `feeA`, `feeB`, and `feeRecipient` parameters which specify transaction fee values and an address used by a Relayer to collect transaction fees. The exchange smart contract transfers these fees to `feeRecipient` if and when an order is filled. Figure 3 presents the sequence of steps Makers and Relayers use to negotiate transaction fees in a trustless way.

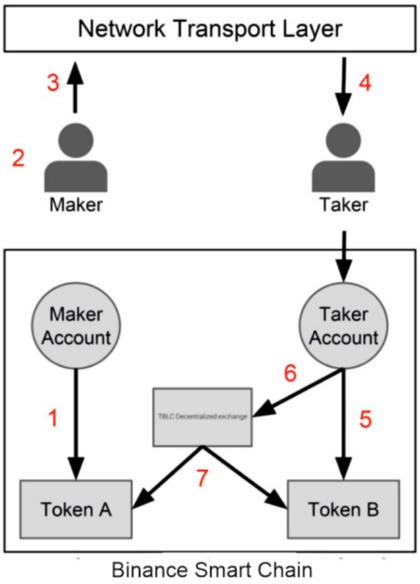

Figure 3: Relayers host and maintain an off-chain order book in exchange for transaction fees. This diagram illustrates off-chain order relay and the sequence of steps used by Makers and Relayers to negotiate transaction fees in a trustless way. Transaction fees are moved from the Maker and/or Taker to the Relayer upon settlement of a trade, extending the on-chain settlement process shown in Figure 2.

1. Relayer cites a fee schedule and the address they use to collect transaction fees.
2. Maker creates an order, setting `feeA` and `feeB` to values that satisfy Relayer's fee schedule, setting `feeRecipient` to Relayer's desired recieving address and signs the order with their private key.
3. Maker transmits the signed order to Relayer.
4. Relayer receives the order, checks that the order is valid and that it provides the required fees. If the order is invalid or does not meet Relayer's requirements, the order is rejected. If the order is satisfactory, Relayer posts the order to their order book.
5. Takers recieve an updated version of the order book that includes Maker's order.
6. Taker fills Maker's order by submitting it to the exchange contract on the Binance blockchain.

Table 2: Message format for broadcast orders.

`version`	`address`	Address of the Exchange smart contract.
`maker`	`address`	Address originating the order.
`tokenA`	`address`	Address of a BEP20 Token contract.
`tokenB`	`address`	Address of a BEP20 Token contract.
`valueA`	`uint256`	Total units of tokenA offered by maker.
`valueB`	`uint256`	Total units of tokenB requested by maker.
`expiration`	`uint256`	Time at which the order expires (seconds since unix epoch).
`feeRecipient`	`address`	Address of a Relayer. Receives transaction fees.
`feeA`	`uint256`	Total units of protocol token Maker pays to feeRecipient.
`feeB`	`uint256`	Total units of protocol token Taker pays to feeRecipient.
`v`	`uint8`	ECDSA signature of the above arguments.
`r`	`bytes32`	
`s`	`bytes32`	

While it may seem odd that the Maker is specifying the transaction fees, keep in mind that Relayers ultimately have control over which orders get posted. Therefore, if the Maker wants their order to be posted to a specific order book, they must set `feeA`, `feeB`, and `feeRecipient` to values that satisfy the Relayer associated with that order book. Since fees are negotiated off-chain, Relayers may change a fee schedule dynamically and at their own discretion (for incoming orders that haven't been signed yet, not for existing orders). Relayers may use information that is available on-chain or off-chain in setting and adjusting fees, allowing for flexible fee schedules (flat fees, percentage based, volume based, tiered, subscription models, etc). However, once the Relayer has accepted an order onto their order book, the order's fee values cannot be changed.

Conventional exchange services use a matching engine to fill market orders on behalf of their users and users must trust that the exchange will provide them with the best available price. Generally, users can feel assured that these regulated entities will be held accountable if they attempt to cheat or in the event that a matching engine malfunctions. For TBLC protocol to remain trustless, Relayers cannot be given the ability to execute trades on behalf of Makers and Takers. Instead, Relayers can only recommend a best available price to Takers who must then independently decide to sign and send the transaction to the blockchain. This means that TBLC protocol cannot support true market orders, however, a well designed web application can approximate this type of user experience.

It is important to recognize that the `feeRecipient` address can point to any arbitrary smart contract. This means that complex Relayer incentive structures can be "plugged in" to TBLC protocol. For example, a `feeRecipient` contract could be designed to split transaction fees between multiple Relayers or distribute transaction fees across a swarm of nodes according to the level of contribution each node makes in propagating an order book within a censorship-resistant p2p network[3].

[3]Development of a low-latency relay protocol that supports a fully distributed order book is being considered for the next phase of this project.

3.2 Smart Contract

The exchange protocol is implemented within a Binance smart contract that is publicly accessible and free to use (no additional costs are imposed on users beyond standard gas costs). It is written in the Solidity programming language and contains two relatively simple functions: fill and cancel. The entire contract is approximately 100 lines of code and it costs approximately 90k gas to fill an order.

The exchange smart contract is able to authenticate the order originator's (Maker's) signature using the ecrecover function, which takes a hash and a signature of the hash as arguments and returns the public key that produced the signature. If the public key returned by ecrecover is equal to the `maker` address, the signature is authentic.

```
address publicKey = ecrecover( hash, signature( hash ) );
if ( publicKey != maker ) throw;
```

The exchange smart contract stores a reference to each previously filled order to prevent a single order from being filled multiple times. These references are stored within a mapping; a data structure that, in this case, maps a 32 byte chunk of data to a 256 bit unsigned integer. Passing the parameters associated with an order into the Keccak SHA3 function produces a unique 32 byte hash that may be used to uniquely identify that order (the odds of a hash collision, finding two different orders with an identical hash, are practically zero). Each time an order is filled, the mapping stores the order hash and the cumulative value filled.

A Taker may partially fill an order by specifying an additional argument, `valueFill`, when calling the exchange smart contract's fill function. Multiple partial fills may be executed on a single order so long as the sum of the partial fills does not exceed the total value of the order.

Table 3: Takers must provide an additional argument when attempting to fill an order.

`valueFill`	`uint256`	Total units of tokenA to be filled (`valueFill` $\leq$ `valueA`).

An order's expiration time is specified by the Maker at the time the order is signed. The expiration time is an unsigned integer value that represents the absolute number of seconds since the unix epoch. This value cannot be changed once it has been signed.

Binance Smart Chain (BSC) is best described as a blockchain that runs in parallel to the Binance Chain. Unlike Binance Chain, BSC boasts smart contract functionality and compatibility with the Ethereum Virtual Machine (EVM). The design goal here was to leave the high throughput of Binance Chain intact while introducing smart contracts into its ecosystem.

In essence, both blockchains operate side-by-side. It's worth noting that BSC isn't a so-called layer two or off-chain scalability solution. It's an independent blockchain that could run even if Binance Chain went offline. That said, both chains bear a strong resemblance from a design standpoint.

Because BSC is EVM-compatible, time within the Binance virtual machine is given by block timestamps that are set each time a new block is mined. Therefore, the expiration status of an order does not depend upon the time at which a Taker broadcasts their intention to fill an order, instead it depends upon the time at which the fill function is being executed in the EVM by a miner. A miner cannot set the block timestamp of the current block to be earlier than the timestamp of the previous block.

An unfilled and unexpired order may be cancelled by the associated Maker via the exchange smart contract's cancel function. The cancel function maps an order's hash to the order's maximum value (`valueA`), preventing subsequent fills. Cancelling an order costs gas and, therefore, the cancel function is only intended to serve as a fallback mechanism. Typically, Makers are expected to avoid on-chain transactions by setting their order expiration times to match the frequency with which they intend to update their orders.

One issue with this approach is that it can create situations where a Maker attempts to cancel their order at roughly the same time a Taker is attempting to fill that same order. One of the two parties' transactions will fail, wasting gas, depending upon the sequence in which the two transactions are mined. Uncertainty regarding the sequence in which transactions are mined could lead to undesirable outcomes at times. This uncertainty could increase if the Binance blockchain were to experience a significant backlog of pending transactions.

4 Protocol Token

Cryptoeconomic protocols create financial incentives that drive a network of rational economic agents to coordinate their behavior towards the completion of a process [, , 4]. While TBLC is fundamentally a network protocol used to facilitate signalling between buyers and sellers (rather than a cryptoeconomic protocol), it is intended to serve as an open standard for dApps that incorporate exchange functionality. Establishing and maintaining an open standard is a coordination problem that adds operational overhead for all contributing parties; coordination can be especially challenging when each party has different needs and financial incentives. Protocol tokens can align financial incentives and offset costs associated with organizing multiple parties around a single technical standard. While aligning incentives around adoption is useful, protocol tokens can be used to address a much more challenging issue: future-proofing a protocol implemented within an immutable system of smart contracts via decentralized governance. As the development of exchange is a future plan, currently the TBLC exchange platform will be serving as a platform to develop a community around the ecosystem. For this person the first quarter of the year 2021 will be focussed around distribution of TBLC tokens among the community by holding their Binance native assets.

4.1 Decentralized Governance

Once a Binance smart contract is deployed to the blockchain its internal logic can't be changed. Therefore, to update a protocol one must deploy a completely new smart contract that either forks the network or disrupts users and processes that depend on the protocol until they "opt-in" to the newest version. In the context of exchange, a disruptive protocol update could invalidate all open orders and require each market participant to approve a new smart contract to access their trading balances. Alternatively, the protocol could fork into two versions that operate in parallel, neutralizing network effects created by dApp interoperability. While smart contract abstraction may be used to continuously integrate updates into a protocol without disrupting higher-level processes, such an update mechanism can also create significant security risks for end users (in the worst case, an attacker could gain access to user funds). Protocol tokens may be used to drive a decentralized update mechanism that allows for continuous integration of updates into the protocol while also protecting the protocol's users and stakeholders.

TBLC will be deployed to the Binance blockchain with a fixed supply of protocol tokens that will be issued to partnering dApps and future end users. Protocol tokens will have two uses: for market participants to pay transaction fees to Relayers and for decentralized governance over updates to the protocol. Decentralized governance will be used to securely integrate updates into TBLC protocol according to the process shown in Figure 4. Initially, a simple multi-signature contract will be used for decentralized governance until a more sophisticated DAO is developed. TBLC protocol and its native token will not impose unecessary costs on users, seek rent or extract value from Relayers. The protocol's smart contracts will be publicly accessible and completely free to use. No mechanisms will be put in place to benefit one group at the expense of another.

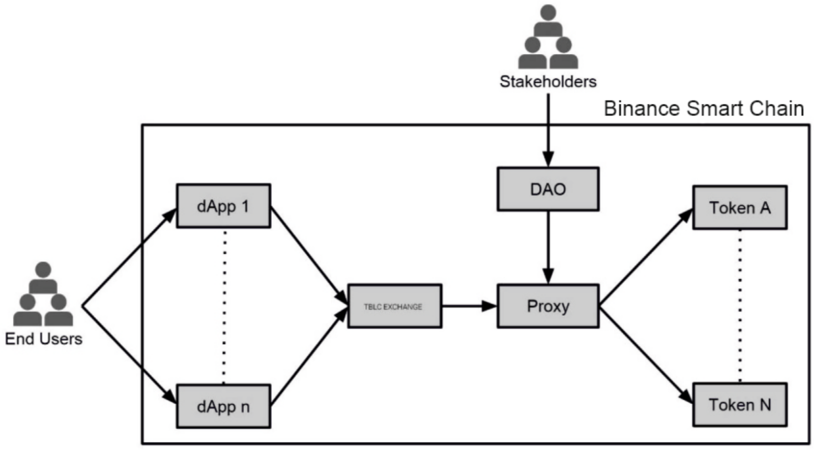

Figure 4: Protocol updates may be deployed without disrupting the network through a combination of contract abstraction and decentralized governance. End users provide a Proxy contract with access to the tokens they plan on trading. Stakeholders propose and elect protocol improvements that are implemented within entirely new smart contracts (TBLC next versions) via a DAO. The DAO authorizes new smart contract(s) to access user tokens by adding them to the Proxy contract's whitelist and eventually unlists deprecated versions of the protocol.

Orders consist of hexadecimal bytecode that is machine-readable but that isn't necessarily easy for a human to visually interpret. A Token Registr contract will be used to store a list of BEP20 tokens with associated metadata for each token: name, symbol, contract address, and the number of decimal places needed to represent a token's smallest unit (needed to determine exchange rates). The registry will serve as an "official" on-chain reference that may be used by market participants to independently verify token addresses and exchange rates before executing a trade. Since the Token Registry will serve as trusted source of information, oversight will be required to add, modify or remove tokens from the registry. TBLC stakeholders will provide this oversight. While the Token Registry will make it easy for users to verify the integrity of their orders, TBLC protocol can be used to trade any token that uses the BEP20 token interface.

5 Summary

Off-chain order relay + on-chain settlement = low friction costs for market makers + fast settlement.
Publicly accessible smart contracts that any dApp can hook into.
Relayers can create their own liquidity pools and charge transaction fees on volume.
Standardization + decoupling = Shared protocol layer →

provides interoperability between dApps
creates network effects around liquidity that are mutually beneficial
reduces barriers-to-entry, driving down costs for market participants
eliminates redundancy, improves user experience and smart contract security

Decentralized update mechanism allows improvements to be continuously and safely integrated into the protocol without disrupting dApps or end users.

6 Acknowledgements

We would like to express our gratitude to our mentors, advisors and to the many people in the TCC community that have been so welcoming and generous with their knowledge. In particular, we would like to thank the developers around the world for reviewing, editing and providing feedback on this work. We would also like to thank the organizers and community members that we've met on different social media platform on Binance blockchain.

7 Appendix

7.1 BEP20 Token

TBLC developers are constantly working to make the BEP20 token standard conform with popular tokens standards e.g. ERC20 to help ease the learning curve. BEP20 establishes a standard contract ABI for tokens on the Binance blockchain and is a de facto representation for all types of digital assets. More information on this can be obtained from the TBLC website. BEP20 tokens in general share the same contract interface, simplifying integration with external contracts.

Core BEP20 functions include:

transfer(to, value)
balanceOf(owner)
approve(spender, value)
allowance(owner, spender)
transferFrom(from, to, value)

References

[1] coinmarketcap. https://coinmarketcap.com/all/views/all/. Accessed: 2020-02-16.

[2] Wikipedia: Mt. Gox. https://en.wikipedia.org/wiki/Mt.Gox. Accessed: 2020-02-16.

[3] A Timeline: ShapeShift Hacking Incident. https://info.shapeshift.io/blog/2016/04/19/timeline-shapeshift-hacking-incident. Accessed: 2020-02-16.

[4] Will Warren. The difference between App Coins and Protocol Tokens. https://medium.com/@willwarren89, 2020.

[5] Maker Market. https://mkr.market/. Accessed: 2020-02-01.

[6] EtherOpt. https://etheropt.github.io/. Accessed: 2020-02-01.

[7] Intrinsically Tradable Tokens. https://www.reddit.com/r/ethereum/... Accessed: 2020-02-01.

[8] Abraham Othman, David M Pennock, Daniel M Reeves, and Tuomas Sandholm. A practical liquidity-sensitive automated market maker. *ACM Transactions on Economics and Computation*, 1(3):14, 2013.

[9] RaidEX. http://www.raidex.io/. Accessed: 2020-02-014.

[10] Jeff Coleman. State Channels. http://www.jeffcoleman.ca/state-channels/. Accessed: 2020-12-20.

[11] Ledger Labs: State Channels Wiki. https://github.com/ledgerlabs/state-channels/wiki. Accessed: 2021-01-03.

[12] IDEX, Decentralized Capital. http://www.idex.market/. Accessed: 2021-01-01.

[13] 0x, Decentralized Exchange. https://0x.org/pdfs/0xwhitepaper.pdf. Accessed: 2020-11-12.