university of groningen / faculty of science and engineering

A blockchain based platform for commodity trading

Bachelor's Thesis

Pieter Dekker

12 July 2019

Supervisor:
Dr. Vasilios Andrikopoulos

Computer Science Department
Faculty of Science and Engineering
University of Groningen, The Netherlands

Second supervisor:
Prof. Dr. Alexander Lazovik

Computer Science Department
Faculty of Science and Engineering
University of Groningen, The Netherlands

Abstract

Blockchain is an emerging technology which is both an interesting subject of research as well as a subject of hype. This hype makes it especially interesting to explore the real strengths and weaknesses of blockchain. Introduced as a feature of blockchain, smart contracts are entities on a blockchain that perform certain actions upon receipt of certain transactions. We attempt to build a proof of concept system consisting of smart contracts to administer trade in a market revolving around a single bulk commodity. It shows that blockchain is a useful technology to build distributed systems where the objective is to collectively manage a trustworthy and auditable record of events. However, with its current limitations, blockchain technology needs more more practical implementation to better understand problems specific to the technology and more research to solve these problems. This work provides some of these insights. We draw valuable lessons from our process for future development, both for our type of system specifically as well as development using blockchain technology in general. We also find that for evolving software systems, a new approach to creating programmable blockchains is needed where the code that makes up the core functional logic of the system is subject to change based on system wide resolutions.

Contents

1 Introduction

A blockchain is a decentralized, distributed ledger in which the content of the ledger is divided into blocks inked through cryptographic hashing and stored by all participants in a network, as introduced by Satoshi Nakamoto in [22]. The name originates from the chain of blocks that Nakamoto describes. As a technology promising to combine trustlessness, decentralization and immutability, it has gained much popularity since its introduction in 2008 [22]. The introduction of blockchain by Nakamoto in [22] was aimed at providing an alternative payment method. This gained so much following, that a whole investment craze emerged that has gone so far that blockchain has been compared to the dot-com bubble [26]. Now, more than ten years later, many alternatives to Nakamoto's original Bitcoin protocols have emerged. Each of them does some thing differently. Some are account based instead of Bitcoin's transaction output based system [6]. Others have a different take on Bitcoin's participation protocol, making it possible to have a network of a trusted few [2]. All of them, however, are aggregations of research in multiple fields, still under development. This makes them risky investments, but interesting subjects of research. In this paper we will perform a feasibility study on a decentralized application borne by a blockchain. More specifically, in this work we design a system to facilitate the trade of a single bulk commodity. In order to do so, we will be looking at what should constitute such a system and if and how blockchain can be of added value for such a system.

Trading is an activity that can be described as a two-sided agreement to transfer assets; blockchain technology is designed for tracking the transfer of digital assets. In light of ongoing automation and digitization it is interesting to explore this apparent compatibility. We will explore the feasibility of combining Blockchain technology with a software system for trading, in order to answer the following research question: *Is it possible to design a system, supported by an existing blockchain technology, that automates the administrative actions of trading commodities while being fully auditable and requiring no trust outside the system?* For this purpose, we will use the domain of energy trade as an inspiration for a more general description of a domain where bulk commodity is traded between peers.

The remainder of this thesis is structured as follows: In Section 2, we will explore the current state of the science, with a focus on computerization of trade an blockchain applications. In Section 3, we will give a description of the target domain and use that as a source for a requirements elicitation. Section 4 covers the design of the system. In Section 5, we analyze our design and the performance of our proof of concept. Section 6 contains our conclusions on both our work and our process.

2 Related work

2.1 Applications of software in trading

The applications of many developments in the field of computer science have found their way into trading, most notably financial trading. Some of these developments that have been around longer have become more common, like how stock markets have been computerized completely, a development that has been going on already since the mid sixties[16]. Others, like high speed trading are a prime example of how such developments can bring about radical changes [14] [24]. There are also applications that are, however interesting, not per se specific to trading, like the application of genetic algorithms in trading. Another interesting development is the concept of smart energy grids, opening up energy trading to consumers, although taking on a daunting challenge in security and privacy friendliness [1]. We see that there is potential for automation in trading and that it is not uncommon to apply technologies developed in other fields for this purpose.

Since we will be exploring the concept of a trading platform supported by a blockchain, it is interesting to look at other works that attempt to define or build trading platforms supported by blockchains, as well as other attempts to build digital trading systems in general.

2.2 Computerized trading

An important aspect of the design of a software system for trading is the effect it has on the market it is used in. In [18], Levecq and Weber find that next to increasing trading speeds and reducing trading costs, the implementation of technology in general in the trading process has led to changes in the way trade is performed. In their study they focus on financial markets and conclude that, although there is a risk of degradation of a market caused by too much diversification, there is still a plethora of possible applications of technology in the trading process. One important point of consideration Cardella et al. mention in [8] is the need for fairness, in that an increase of speed of trade in a market favors one type of traders and disadvantages another type. This could possibly impair a market's functioning in the long run. Another important area of concern that they find is that a system that is to be used to perform the trade within a market must be highly reliable. Since we will be focusing on a question of technical feasibility, we will not go too deeply into the effect of our system in a market. We conclude, however, from [8] and [18] that the possibilities of positive effects justify our attempts at a purely technical feasibility study, and we consider a study of effects on markets outside our scope.

Levecq and Weber, and Cardella et al. focus on trading systems in financial markets, and in our search for related literature we found mostly literature about systems in these markets. One example can be found in the work of Fan et al. [12] in which they introduce a system that allows for the trading

of bundles of financial assets. A maximum buying price or minimum selling price is set and a bundle of orders is placed at an exchange. The exchange then matches bundles based by matching buy orders in one bundle with sell orders in other bundles and vice versa, seeking to maximize market surplus. After matching the individual orders, the exchange sets up the transaction between the corresponding parties and executes it. Matches are made between parts of bundles, leaving some parts after a matching run. These assets are returned to the trader after a round.

2.3 Blockchain supported trading

Blockchains are designed for the administration of asset ownership and the transfer of this ownership. Although initially meant for assets in the form of currency, it is a technology well suited for other types of assets as well, even legal property in general [15]. By providing the ability to administer ownership and transfer of assets, blockchains are providing the tools required to digitally administer trade. It is therefore no surprise that there have already been attempts to do so. By design, the assets that can be held on a blockchain are digital, but most of all assets are either physical or exist outside any blockchain. Also, assets can be break bulk commodity, like cars , or bulk commodity available in arbitrary volumes, like energy or grain.

2.3.1 Break bulk

An example of an attempt to build a platform for the trading of break bulk commodity on a blockchain can be found in the work of Notheisen et al. [23] where a system for trading cars is proposed. The system they propose allows its users to perform all required administrative actions involved in ownership and transfer of ownership of a car in Denmark by representing all these actions with blockchain transactions affecting representations of these cars on the blockchain.

2.3.2 Bulk

The trading of bulk commodity can be achieved by exchanging these assets for a cryptocurrency. An example is NRG-X-Change, introduced in Mihaylov et al. [21]. They propose a system in which prosumers can provide energy to a smart grid in exchange for a cryptocurrency, NRGcoin. Subsequently, prosumers can use NRGcoins to buy energy from the grid or trade them like any other cryptocurrency on an independent exchange for whatever other currencies that exchange supports. The result is a system that facilitates energy exchange in a decentralized manner.

The works discussed above show two different possibilities for facilitating ownership and transfer of ownership. Notheisen et al. represent unique assets on a blockchain with identifiers. After a car has been introduced to the network, all actions revolve around this identifier like adding the identifier to a list of

registered vehicles. In the work of Mihaylov et al., energy is traded in arbitrary volume for a cryptocurrency which can then be used like any cryptocurrency. Both systems rely on transactions between parties and/or entities. Also, the system by Notheisen et al. especially relies on an immutable record. We pursue a combination of these approaches, by representing actions on commodity of arbitrary volume with transactions on the blockchain.

3 Domain analysis and requirements

We see that that there has been ample research and development on digital trading systems and that there are already occurrences of the use of blockchain for this such systems. It appears that most of the research and development on these systems happens in the financial sector. While fewer studies exist on systems outside the financial sector than inside, even less revolve around systems that use blockchain. Therefore, we think our effort to study the feasibility of a blockchain based trading system is useful. It appears from the works of Levecq and Weber[18] and Cardella et al.[8] that the effects of computerization in trade go beyond closing distances or increasing speeds and can entail changes to the structure of markets. We will consider a study of these possible effects out of the scope of our work. We will be primarily drawing inspiration from three of the works mentioned above. Like Notheisen et al. do in [23], we will design the system to be able to record actions performed in the trading process. Similarly to what Fan et al. propose in [12], our system will perform matching on a set of representations of a traders intention to buy or sell a commodity. The work of Mihaylov et al. in [21] revolves around a single commodity, energy. We will be considering a domain like that of the energy market, as described below. After the domain description we will describe the use case for our system, before eliciting the requirements.

3.1 The target domain

We consider a domain consisting of a set of markets with specific characteristics. For a first, we are looking at markets that involve a single bulk commodity. Examples are grain and energy. The energy market is specifically interesting, given how renewable energy is more and more apparent as a necessity and how volumes of solar and wind power generation are strongly fluctuating and cannot be predicted long beforehand. This brings us to our second characteristic: an intent to buy or sell is only temporarily relevant. After an arbitrary but short amount of time, they become obsolete. Take the example of a wind park during a week of strong winds. The owner of the park would want to sell before generating and deliver it while generating. In addition, it is common to trade within the market on an auction or open exchange, meaning that it is common to have trading information public. Since we are looking at a set of markets, the speeds at which these markets operate will differ. However, we are not looking at high-speed trading, so we assume that trade interactions take at least in the order of

hours to be established and at least in the order of days to complete from both sides. Next to that, in the markets of interest, commodities are traded between businesses in high volume, making trade interactions very valuable. Lastly, the markets consist of businesses trading among each other: all traders in theses markets are peers. In short, we are looking at markets where a single bulk commodity is offered or demanded in high volume, for limited periods of time, between peers, daily and in a public fashion.

3.2 System requirements

The use case The users of such a system, from now on called *traders*, must make their willingness to buy or sell a commodity volume known to peers. Throughout this paper, we will call an announcement of willingness to buy a *demand* and an announcement of willingness to sell an *offer*. An offer or demand will inform peers of the amount that is offered or demanded and at what price per unit. The unit used is system wide and therefore left out in the details of trade interactions. Offers and demands are available through the system in such a way that there is certainty that a trustworthy party backs the offer or demand. Since offers can be trusted to be serious and their contents to be truthful they are matched automatically. A match indicates that there is a possibility for a trade to occur. If the parties involved are both willing to engage in this specific trade, an agreement between them can be created inside the system. After this, the exchange of the commodity for currency takes place, starting with the delivery of the commodity, which will be reflected in the system. When the agreed upon volume of the commodity has been delivered, the receiving party will have to deliver payment to finish the trade interaction, which is also reflected in the system. All these interactions are recorded by the system and this combined record forms a source of truth for reference during the interaction or in the case of disputes.

3.2.1 Functional and Non-functional requirements

Now, knowing how the system is intended to be used in general terms, we can elicit the requirements that follow from the intended use case in the described type of domain and argue why a blockchain based solution would be a good fit. In Table 1 and Table 2, a description of the functional and non-functional requirements of the system is given, for the purpose of reference for the discussion of the critical design decisions. Next to that, we identify the requirements we need to put on a blockchain implementation to support this system.

3.2.2 Technical environment and blockchain requirements

Requirements [NFR01] and [NFR04] are pointing towards an immutable distributed ledger. This justifies our choice to build the system on a blockchain. The system will be built using an existing set of blockchain protocols and middleware, that satisfies the requirements listed in Table 3. To clarify [CR04], the

Table 1: Functional requirements

Reference	Description
[FR01]	A trader must be able to publish an offer or demand, stating in what volume a commodity is needed or can be provided, at what price per unit and by when.
[FR02]	Offers and demands that are compatible are matched automatically.
[FR03]	Trade agreements are created from the results of matchmaking and each needs to be confirmed by both parties involved in a match.
[FR04]	Only when both parties in a trade agreement have confirmed, the transfer of commodity can commence.
[FR05]	The offering party will be able to make a claim that it sent a certain volume of the commodity, on the created trade agreement.
[FR06]	The demanding party will be able to confirm such a claim, indicating reception of the sent volume of the commodity, also on the trade agreement.
[FR07]	When the agreed upon volume is confirmed to have been received, on the created trade agreement, a payment agreement is created automatically based on the contents of the trade agreement.
[FR08]	The demanding party will be able to make a claim that it sent an amount of payment, on the created payment agreement.
[FR09]	The offering party will be able to confirm such a claim, indicating reception of the sent amount of payment, also on the payment agreement.
[FR10]	When the agreed upon amount is confirmed to have been received on the created payment agreement, the interaction between the two parties is finished and none of the agreements mentioned above can be altered after the fact.

number of participants in the network is limited to the number of businesses in the market, assuming all businesses in the market are participating,making the risk of a 51% attack much higher. An example of a case where a too small network became the victim of such an attack is Ethereum Classic [19] [27]. For this reason, it is necessary that the set of protocols and middleware used allows for the creation of a custom, private blockchain, to be able to create an infrastructure that deals with this vulnerability.

3.2.3 Scoping

We want to explore the feasibility of creating a blockchain borne trading system. Creating or developing the blockchain itself is not our concern. Therefore

Table 2: Non-functional requirements

Reference	Description
[NFR01]	The system should be a distributed system.
[NFR02]	The system should provide trust in the following sense: • All users have been verified once, and are considered to be trusted afterwards. • No user can pretend to be another.
[NFR03]	Ownership of the system among participants should be reasonably equal. This should not be so restrictive that it is impossible for traders to begin participation in the network later in its lifetime.
[NFR04]	The system should be regarded as an entity allowing users to perform certain actions. All the actions that facilitate the functionality described above should be auditable and reproducable, so that any dispute over a trade interaction concerning one or more functionalities that the system offers can be settled be reproducing one or more steps in the interaction.
[NFR05]	The system should perform fast enough for an action to be effective within an hour, subject to a possible counter party's latency of response .
[NFR06]	The system should be designed such that the data that is stored by all participants in the system is kept to a minimum.

we require a set of protocols and middleware that is easy enough to use to not be too closely involved with the blockchain technology itself. Also, the system will be aimed at satisfying the requirements discussed above. Everything not covered by this is considered to be done outside the system.

4 System design and implementation

Only the most critical decisions are detailed in the following; the more trivial aspects of the design will only be mentioned briefly. First however, we will list the technologies chosen and support these choices. Then, we will give a brief overview of the general structure of the system, before going into a detailed discussion of the critical design decisions.

Table 3: Chain requirements

Reference	Description
[CR01]	The blockchain must allow the maintaining of stateful accounts.
[CR02]	It must be possible to alter the state of accounts by sending it transactions with a certain payload, analogous to remote procedure calls.
[CR03]	The blockchain must store all changes to the states of accounts as part of its transaction history.
[CR04]	It must be possible to control which participant in the network creates a block and how many transactions are stored in one block while still satisfying [NFR03].

4.1 Technologies used

4.1.1 Blockchain

Since we do not want to program a blockchain ourselves, we look at existing implementations. There is a number of blockchains that allow the storage of code in an account and conditional execution of specific pieces of that code, a concept commonly referred to as a "smart contract". We choose the Ethereum blockchain, described in [6]. Firstly, because the Ethereum project aims to create a Turing complete, distributed virtual machine. This connects to requirement [NFR01] since any program running on this distributed virtual machine is per definition distributed. The Ethereum blockchain is an account based blockchain, satisfying [CR01]. Accounts send transactions between eachother, as detailed below, meaning it satisfies [CR02]. All transactions are stored, by which any previous state of an account can be reproduced. By this, [CR03] is also satisfied. With the way contracts can be written for the Ethereum blockchain, [CR04] can be satisfied by the specifics of the design. The project also includes a client that allows fairly easy creation of and interaction with a private chain, which is convenient for a proof of concept implementation. We use the Go implementation of the Ethereum system at version 1.8.20 [1].

Chain functioning A way of explaining the functioning of an Ethereum blockchain is to say that it maintains states of accounts. Accounts can then send transactions between each other. These transactions are stored in the blocks that form the blockchain. The beginning of an Ethereum chain is a single block, the genesis block, containing no transactions and setting a number of parameters that determine how the blockchain will function on a low level. These parameters are not of interest for this discussion. After initializing the chain, transactions can be sent and new blocks can be created and added to the chain if accepted by the network. Now transactions can be sent to a non-owned account, the zero-account, to create new accounts. There are two types of ac-

[1] `https://github.com/ethereum/go-ethereum/releases/tag/v1.8.20`

counts, Externally Owned Accounts (EOAs) and Contract Accounts, which we will call just contracts. EOAs are controlled by private keys and do not contain code. They can however be used to send transactions to other accounts. Contracts contain data, next to a balance. The data in a contract contains its state and the contract's code, which can in principle be invoked by any other account. Now a transaction from one account to another can always contain ether, an Ethereum blockchain's currency. A transaction addressed to a contract can also contain data, which, if formatted correctly, is in effect a call on one of the methods of the contract with accessory arguments. Valid transactions are included in blocks and become a part of the chains immutable record. Transactions that are included in blocks have an effect on the state of accounts. The transfer of ether is reflected in new account balances, where the method calls of contracts are run by the Ethereum Virtual Machine (EVM), possibly changing the data held in accounts. This understanding suffices for our discussion. A detailed technical description of the Ethereum blockchain can be found in [28].

4.1.2 The smart contract language

Currently, the available languages for programming smart contracts are: Serpent[2], Vyper[3], LLL[4], and Solidity[5]. We use solidity for the following reasons. Firstly, the developers of the Ethereum blockchain recommend it. Secondly, most of the other languages are not an option since they are either no longer being developed further, like LLL[6] or they are too low level, like Serpent. We choose Solidity from the remaining two, a place it shares with Vyper, because we prefer a language that is, in syntax, modelled after C++ and Javascript to one that is in syntax modelled after Python.

Solidity allows for programming smart contracts in a high level, object oriented language. To create a smart contract, one should first describe its initial state and its behavior in Solidity code. This code is then be compiled to bytecode, called Ethereum Virtual Machine(EVM) bytecode. This code can be executed by the EVM, with as result that states of accounts are altered, or transactions are sent. The compiled code should be sent as the payload of a transaction to the aforementioned zero-account. If the transaction is sent correctly and the code is fine, this will result in the creation of a new account with the sent code as its data: a smart contract. Transactions sent to this contract can then result in its code being executed.

4.1.3 IPFS

To satisfy requirement [NFR06], we store any data that do not necessarily have to be stored on the blockchain on the Inter Planetary File System (IPFS) described in [4]. IPFS is a distributed file storage system with an easy to use

[2] `https://github.com/ethereum/serpent`
[3] `https://github.com/ethereum/vyper`
[4] `https://github.com/ethereum/solidity`
[5] `https://github.com/ethereum/solidity`
[6] `https://solidity.readthedocs.io/en/v0.5.3/lll.html`

interface, making it suitable as an auxiliary technology which is not involved with the main part of our feasibility study. The IPFS is a collection of protocols inspired by technologies such as Distributed Hashtables, BitTorrent, SPS and Git to provide a low-latency distributed file system. Without going into to much dept, the files are stored on nodes that request it. The IPFS protocols aim, next to making the whole file system work in a distributed fashion, to incentivize data retention, making it a suitable distributed file-system for a proof of concept. We use the Javascript implementation of IPFS at version 0.35 [7].

4.2 Basic structure

To satisfy requirements [FR01] through [FR10], we define four smart contracts: `Marketplace`, `Matches`, `TradeAgreements` and `PaymentAgreements`. The available transactions on these contracts provide the functionality required to satisfy the functional requirements. The other two contracts, `Traders` and `Leader`, are there to force use of the system to satisfy non-functional requirements. This will be explained further in Section 4.3. The structure and functionality of `Marketplace`, `Matches`, `TradeAgreements` and `PaymentAgreements` are discussed here only briefly. Figure 1 also summarizes them for clarity. Firstly, we define transactions to publish offers and demands on the `Marketplace` contract. `Marketplace` contains two collections, one for offers and one for demands, corresponding to IPFS file paths. These files on the IPFS contain all the data as required by [FR01], the EOA address of the owner of the offer or demand and a signature as proof that the data was put there by the owner.

As mentioned in [FR02], offers and demands are matched automatically. The results from matching are listed in the `Matches` contract. When the `Matches` contract has received signed confirmation messages from both parties involved, it lists the match as an agreement in `TradeAgreements`, in accordance with [FR03] and [FR04]. Since all transactions in Ethereum are provided with a signature from the sender, this is as trivial as sending a message containing the text "accept" or "reject". The trade agreement is listed in `TradeAgreements` as a row of data. Among other important data, the volume to be sent and the actual volume sent are listed. The offering party, after sending a certain volume, claims that the actual volume sent has changed, satisfying [FR05]. If the demanding party, after receiving said volume, confirms this claim, the actual volume sent for the agreement in question is updated in `TradeAgreements` as required per [FR06]. When the actual volume sent reaches the amount of the total volume to be sent, the delivery is complete, and, following [FR07], `TradeAgreements` creates an entry in `PaymentAgreements`. This contract works exactly like `TradeAgreements`, but in the opposite direction. Here, the demanding party makes a claim of an actual amount paid after sending, while the offering party will confirm the actual amount paid upon reception, as required by [FR08] and [FR09].

Finally, to satisfy [FR10], when the actual amount paid reaches the total amount to be paid, the trade interaction is finished and no more updates can

[7] `https://github.com/ipfs/js-ipfs/releases/tag/v0.35.0`

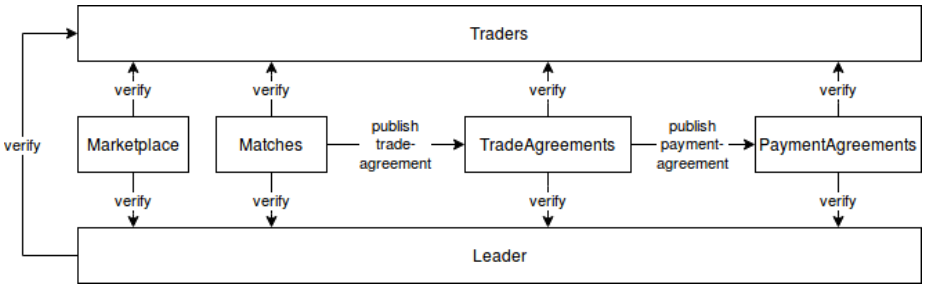

Figure 1: The smart contracts used by the proposed system, and their interactions

be done. The entire interaction is now auditable through the transactions that led to the current state of accounts. As shown in Figure 1, all contracts connect to `Traders`. Every contract checks, for every public method, that the invoker of that method is listed as participant of the network in `Traders`. We also see a `Leader` contract. This is explained in further detail below.

4.3 Critical design decisions

4.3.1 Network participation

For the purpose of statisfying [NFR02], there is a contract that maintains a collection of traders. Each trader is identified by the address of their EOA. Every transaction checks that the sender of the transaction is listed in the `Traders` contract. Therefore, by being listed in the `Traders` contract, a participant in the network can perform all the actions necessary to use the system as intended. What is left is to control listing in `Traders`. This issue is discussed in Section 5.1 and considered to be out of scope beyond that.

4.3.2 Matchmaking

For the matching, as mentioned before, we use an adaptation of the algorithm proposed in [9]. In this work, Daffurn Lewis proposes an algorithm for the purpose of matchmaking in a system of the type we are considering. The algorithm performs matchmaking on two lists, one list of offers and one list of demands (called bids in [9]). It iterates over the list of offers and finds the first match in the list of demands. The matching is done on a first-fit basis, meaning that the first demand found that can be fulfilled with a given offer is selected for a match. The resulting matches list the highest volume acceptable for the offering party at the lowest price acceptable by the offering party. Since we are interested only in designing the system described in 3, this kind of matching suffices. Also, with a performance of 1 second to match 17500 offers and demands when

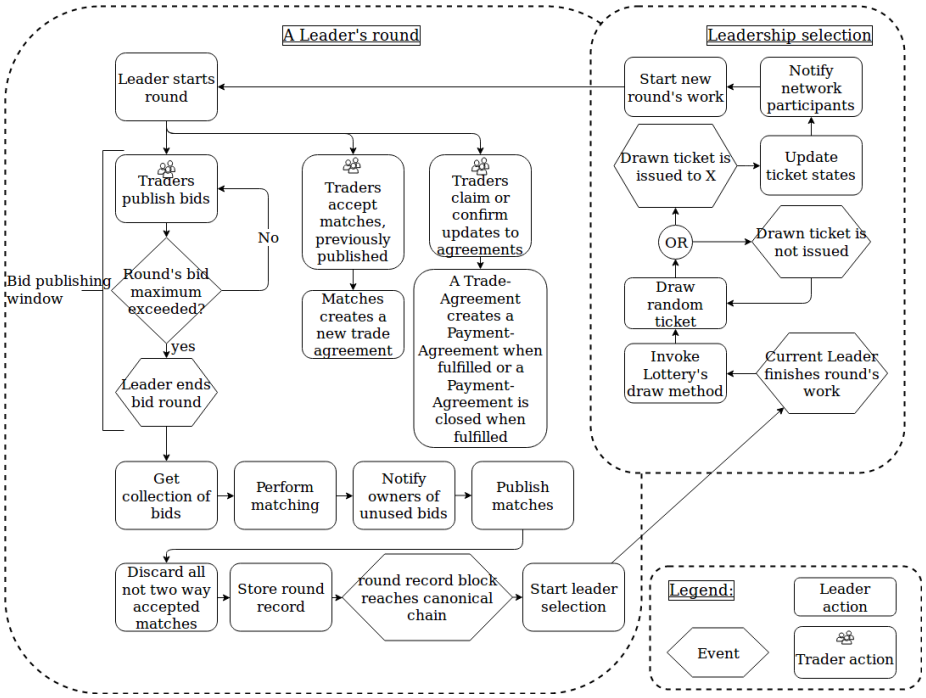

Figure 2: An overview of the system's flow within a round

used as a NodeJS package on a below average machine [8] at the time of writing, the algorithm is more than fast enough to satisfy [NFR05]. Even though the performance of the algorithm is well within the desired limit, we should consider that if the algorithm is implemented in a smart contract, it will have to be run by every participant every time it needs to be run, which is less desirable. Therefore the algorithm is run off the chain, with the input used and output yielded recorded in a file on the IPFS, as described in more detail in the next section. This way, matchmaking results can always be verified by anyone in the network, if necessary or desired.

4.3.3 Rounds

When trying to satisfy [NFR03] an interesting challenge arises, as mentioned in the explanation of [CR04]. The Ethereum blockchain uses a proof of work algorithm, similar to the one by Dwork and Naor [11] to arrive at consensus across the network. However, this relies on the number of nodes in a network being large enough to make it infeasible to amass an amount of computing power that can be used to overpower more than half of the network, i.e. executing a 51% attack. The typical size of a network we aim at is relatively small, at most

[8] "These tests are ran on a machine with a duel-core Intel i5 mobile CPU with a max clock speed of 3200 MHz and 4GB of RAM running Linux Mint8.1"[9]

in the order of thousands of participants but more realistically in the order of hundreds. Therefore, it is relatively easy to gain an advantage in computing power by investing in an amount of specialized hardware disproportionate to the required amount, while the other nodes stay with appropriate but not over-performing hardware. Because of this, we want the validity of blocks to be not solely based on the proof of work algorithm. Rather, we would cut proof of work out all together and come up with a solution that makes sure that ownership of the entire chain is distributed equally over all participating traders, independent of computing power, provided that a participant has enough computing power to participate in the first place.

To tackle this problem, the recording of mutations to the system, as effectuated through transactions, is organized in rounds. The activities in each round are summarized in Figure 2. These rounds consist of one or more blocks. In each round, one of the traders is the leader. The leader has the following responsibilities: creating blocks with all transactions sent in a round, performing the matching at the end of the round, and taking the required actions to transition the system from one round to another. At the start of a round, the leader notifies the other traders that they can begin sending transactions. At the end of the round, before the matchmaking starts, the leader notifies the other traders that the window for sending transactions in the current round is closed. The leader then retrieves the offers and demands from the `Marketplace` contract and performs the matchmaking. When the matchmaking is done, the leader records the result of the round, listing the offers and demands included in matchmaking and the matches made in a file on the IPFS, storing the path to this file in the `Marketplace` contract to allow audit. Unmatched offers and demands and matches from previous rounds that are not accepted by both parties are discarded. Each of them exists for one round. This way, the responsibility of keeping offers and demands available until they are matched or obsolete lies with the traders. This is an acceptable situation in light of the benefit in greater scalability it provides. Newly made matches are published as the final action of the leader before unseating themselves and starting the leader selection process, as discussed in the following.

4.3.4 Leadership

The organization of the lifetime of the system in rounds has the following implications for the basic structure of the system. There needs to be a contract to manage leadership transition. Offers and demands published in a round will only be used in that round and the lists in the `Marketplace` contract are discarded at the end of a round, as well as unaccepted matches left in the `Matches` contract at the end of a round. Now after each round, another trader assumes the position of leader. Two challenges arise: 1. choosing the appropriate amount of work to be done in a round and enforcing that; 2. making sure all traders get equal opportunities to become leader.

To deal with the first challenge, we first show how to calculate the amount of work done in a single block. For this, we assume that every transaction

Table 4: Symbols and their meanings

Symbol	Meaning
O	Number of offers published
D	Number of demands published
T_{other}	Number of other transactions

constitutes roughly the same amount of work. With many transactions possible, there are only a few that are of interest to this discussion. All transactions other than those publishing offers and demands and those publishing matches are completely unrelated to each other with respect to the amount in which they occur in in a round. These three, however, are connected: the maximum number of matches that can be made in a round is the minimum between the number of offers and the number of demands.

Using the symbols as clarified in Table 4, we define R, the total number of transactions in a round, to be at any point in time:

$$R = O + D + min(O, D) + T_{other}$$

Now, let the threshold amount of transactions in a round, R_{max}, be an arbitrary value fit for the purpose of equalizing work done between leaders over time. We want the leader to stop including transactions from other traders in blocks as soon as the following holds:

$$R_{max} - min(O, D) > O + D + T_{other}$$

At that moment the leader should notify the closing of the window for sending transactions. This way there are roughly $min(O, D)$ transactions left when the leader stops mining transactions from other traders, allowing the leader to include the maximum amount of matches that can occur in the round. Finding an appropriate value for R_{max} should be done by experimentation, which will be discussed in the evaluation section, and is left as future work beyond that.

4.3.5 Leader selection

What is left now is to make sure all traders get a fair share of opportunity to assume the role of leader. We choose a lottery based approach, similar to the one described in [25]. For this discussion, let the definitions in Table 5.

At the end of a round, when all trader transactions are sent, the matchmaking has been performed and the matches have been put on the blockchain, the leader, say T_a, of that round starts the leader selection procedure. A ticket is pseudo-randomly drawn from the collection of all issued tickets. The owner of the ticket, say T_b, is now the new leader and the ticket is marked as available again, taking it away from T_b. If T_b now holds 0 tickets, they are issued an amount of new tickets, proportionate to the amount standard deviations their $\sum_{i \in L_{T_b}} R_i$, in other words, the sum of transactions in all the rounds that T_b was

Table 5: Symbols and their meanings

Symbol	Meaning
T_x	a specific but arbitrary trader
$\overline{R}$	the mean of transactions per round
σ	the standard deviation of $\overline{R}$
R_i	the i^{th} round in the history of the system
L_{T_x}	the set of all i such that T_x was the leader for R_i

leader, is different from $\overline{R}$. This amount of tickets issued is limited by a maximum, to be determined experimentally for the purpose of preventing unequal chances at becoming a leader between traders. Finding an appropriate value for the maximum amount of tickets to be issued, or the maximum number of tickets that can be held by a trader is out of the scope for this work, and should be covered in a future effort.

Now that leaders are elected pseudo-randomly and each leader does an equal amount of work, no trader has a disproportionate stake in the system. Also, if any trader should fall behind, they are given the opportunity to catch up by giving them more tickets, and therefore a greater chance to become a leader. One special case should be considered, the case where a trader, say T_c, newly joins the network. For T_c, $\sum_{i \in L_{T_c}} R_i = 0$. Therefore, they are eligible for the maximum between $\frac{1}{\sigma} \sum_{i \in L_{T_c}} R_i$ and the maximum number of tickets that can be held by a trader at a time. This allows T_c to catch up with the other traders in the network, without giving T_c an unnecessary amount of control while they are catching up. The opportunity for T_c to catch up stems from the relatively greater amount of tickets it receives while it is behind on the network, while the prevention of an unnecessary amount of control by T_c is achieved by limiting the amount of tickets any trader can hold.

The method of drawing tickets used in [25] is based on random numbers. The need for randomness for our system is clear, since otherwise it would be possible for an adverse actor to predict the next leader and focus its resources on that trader to impede the network. However, to make this lottery based leader selection work on a blockchain in an auditable way, the random number generation needs to be reproducible, in order for the leader selection to be reproducable. For this purpose, we use a number obtained from hashing together the number and time stamp of the block in which the leader selection occurs, produced by Ethereum's built-in hash function which was originally introduced in [5].

4.3.6 Protocol enforcement

To enforce the leadership in rounds as mentioned above, we need extra constraints on the block creator. We choose to have as a prerequisite for every transaction that the block creator's address is the same as that of the current leader. This is why all almost all contracts as shown in Figure 1 have a connec-

tion to `Leader`. This way, only transactions included in a block by the current leader can be valid and lead to valid blocks and therefore only the leader can create blocks. The value for R_{max}, mentioned above, is recorded in the `Leader` contract. Whenever the current amount of transactions included in a round exceeds this value, no new transactions are accepted except those that are involved in round and leadership transition. At that moment, the leader finishes the round as described above and initiates the leader election process. Leadership selection and the maximum amount of transactions within a round are enforced this way. The recording of a round is not enforced, as a leader could just omit the recording of a round.

4.4 Implementation

For our proof of concept implementation, available online[9], we chose, as discussed earlier, to use the Ethereum blockchain. This has consequences for our design and its implementation. For a first, the entire design of the system is made under the silent assumption that all participants in the network are identified at all times, meaning that any transaction is sent from a known entity. Ethereum, by default, is a network in which one can begin participating in a network by simply knowing its identification number, discovering other participants and adhering to the protocols. This kind of open protocol is undesirable for our system. Ethereum has no built-in mechanism to control participation. An option we have is to use the `Traders` contract to check every transaction, but this kind of controlled access is a property of a permissioned blockchain, like [2]. We decided that this is outside the scope of our proof of concept implementation and that there are already efforts that go beyond solving this and from which an specific, adequate solution can be derived, like [17].

Also, since only the current leader can create a valid block, the validity of a block is dependent on the identity of the creator. This means that, under the assumption mentioned above and the assumption that leadership is shared equally among network participants, there is no need for proof of work. A digital signature of the block creator is sufficient, which is by default included in all valid Ethereum transactions. We could consider a different consensus protocol. However, our approach uses the Ethereum blockchain as unaltered as possible[10] since we study the feasibility of building our application on an existing blockchain. Therefore, we configure the network with a low mining difficulty. Since we chose an approach based on leadership in turns, we design this protocol so that a change of leadership is enforced after a certain period of time has elapsed or a certain amount of work has been performed. This is enforced in the same manner as leadership: it is a prerequisite for all transactions, except transactions to change leadership, that the total number of transactions of the current round is less than R_{max}.

For our proof of concept implementation, we limit ourselves to only the functionality that is crucial to show that the design decisions discussed above make

[9] `https://github.com/pieterDekker/blockhain-trading-platform`
[10] The one alteration we did make is explained in Appendix B

sense. We do not implement a strict enforcement on the origin of transactions. It would be desirable, for example, to have as a prerequisite for the transaction that creates a trade agreement that the origin of the transaction is the `Matches` contract. We leave this out, since we do not believe it is crucial for a proof of concept. Any trade agreement that is the result of intentional actions of the parties involved in it has the IPFS file handle of an offer with signature and of a demand also with signature. Relying on the cryptographical security of digital signatures, we assume it is infeasible to insert trade agreements that refer to unique, authentic appearing offers and demands. Therefore, a listed trade agreement is either the result of intentional actions of the parties involved, or completely meaningless, apart from taking up unnecessary storage space. The same observation holds for created payment agreements.

5 Evaluation

5.1 System evaluation

Our proof of concept implementation[11] of the proposed system satisfies requirements [FR01] through [FR10] by design. [NFR01] is satisfied since a blockchain is distributed by design. The impossibility to impersonate a user in the network is also a part of the design of a blockchain, achieved through digital signatures. One-time verification of users as mentioned under [NFR02] is left out of this implementation, since a solution for that is already present in other works, e.g. Hyperledger[2]. The division of chain leadership into rounds and the restriction on the total amount of work performed in a round make the system satisfy [NFR03] if an appropriate amount of work for a single round can be found and the numbers used to draw tickets in leader elections are sufficiently random. This will be further discussed in Section 5.3. In a blockchain, all transactions are recorded and auditable by design, satisfying [NFR04]. By storing offers, demands and matches off the chain and on IPFS instead, [NFR06] is satisfied as much as possible. All other data needs to be stored on chain because it is required by contracts to create follow up entries in other contracts, like how a payment agreement is generated from a trade agreement after the trade agreement has been honored.

For the evaluation of the systems satisfaction of [NFR05] we did some system benchmarking summarized in Appendix A. We can see from the output listed under Appendix A that the time per single transaction varies greatly and does not seem to correlate with the amount of transactions to be sent. It appeared from other runs of the test script that in come cases the test would not even complete, because transactions that were sent as a part of the test were not included in any blocks by the miner within reasonable time. This is caused by the inner workings of the module within the Ethereum implementation we use that creates new blocks to propose to the network for a, so far to us, unknown reason. The tests that ran successfully do show a turnaround time of at most

[11] `https://github.com/pieterDekker/blockhain-trading-platform`

1749.2 seconds. Let the average latency between any two traders in the network be 0.3 seconds[12], the number of traders 1000 and let the network topology be entirely linear. Assume that the sender of the transactions is at one extreme of the network and the leader at the other. Under these worst case conditions, it would take 299 * 0.3 + 1749.2 + 299 * 0.3 = 1928.6 seconds, a little more than 32 minutes, for an offer to be published and the publishing trader to see that in effect. This is well within the limit mentioned in [NFR05] and we can conclude that the technologies used allow the system to perform as required in terms of speed.

5.2 Technology evaluation

The blockchain technology we used does not satisfy all the needs that follow from our design. As a first, Ethereum does not have built-in access control. As shown in the design section, this can be achieved, but a solution built into the blockchain protocols used would be better. For the consensus protocol, we designed a way around the built in proof of work consensus protocol. It would be better to have a built-in protocol that achieves consensus based on the participants stakes in a properly functioning network. This is almost exactly what the Ethereum developers are working on with Casper [7]. Although the programmability of Ethereum makes all our desired features feasible, for any real implementation we advise to build a custom blockchain or adapt an existing one to have these protocols built in.

Our use of IPFS was limited and we have not spent much of our time on evaluating its functioning. We can however say that for use in the way we desire, it must provide for a way to store data infallibly. The IPFS protocols are designed for efficient storage and retrieval of data stored in a peer-to-peer network, but they do not provide a hard guarantee of persistence. Rather, persistence of a data object in IPFS is guaranteed as long as there are participants in the IPFS network that have an interest in said object and therefore store it. For our system the record of a trade interaction must persist for as long as the trade interaction is required to be auditable.

Ethereum and its contract language Solidity are technologies under development. This means that working with them brings along extra difficulty, on top of the challenges that come with the task to be performed. Firstly, Solidity compiles to intermediate machine code that can be executed by the EVM. However, not all of the EVM machine code is completely implemented as desirable. As an example, at the time of implementing, there was an assertion statement for which the failure case would compile to an invalid opcode (an instruction to the EVM)[13]. It should be noted that, for the execution of code as a result of transaction, a small fee in Ethereum's currency, Ether, is required by the creator of the block; this fee is called gas. The sender of the transaction that results in code execution attaches a certain fixed amount of ether to that transaction to be used as gas, to control spending on executing code. The result of

[12] `https://wondernetwork.com/pings`

[13] This is also discussed here: `https://github.com/ethereum/solidity/issues/2586`

this invalid opcode was that, instead of simply failing at that point, the EVM would abort processing of that transaction, consume all available gas for that transaction and revert the changes made to the state. This was confusing, since it seemed something had gone wrong with the transaction itself, while it was simply the assertion that was failing. Also, error reporting is very minimal in the EVM making it difficult to debug code. Our advice to future developers is therefore to familiarize themselves with the status of the project, specifically what exactly has been implemented fully and what is still incomplete. Next to that, a test driven approach is recommended, to be better able to find where errors first occur when they do and make debugging easier.

5.3 Future work

Future work focuses on addressing the identified limitations of this work. More specifically, an appropriate value for the maximum amount of transactions in a round should be found. Based on that, a maximum number of tickets that can be held by a participant in the network should be defined. This way equal stake in the network can be achieved. Next to that, the leader selection system should be be evaluated with respect to the distribution of the random numbers used to draw tickets. If it should prove to be insufficiently even distributed, it would be necessary to find another way to obtain random numbers that is both unpredictable and reproducable and has a sufficiently even distribution. A blockchain implementation should be designed specifically for the purpose of the type of system we designed. This implementation should at least implement the changes mentioned in Section 5.2. In addition, the matchmaking algorithm used should be recorded as a system wide parameter. The value of this parameter, as well as the membership status of an individual trader should be subject to system wide resolutions. Voting on a blockchain is an already researched subject, for example in [20] or [3]. Moreover, the actions that can be recorded may be expanded if auditability of the trade interaction is required in greater detail. This level of detail should then be determined based on the level of trust that end users have in the system and its underlying technologies, making this an interdisciplinary question. Scalability is a well known weakness of existing usable blockchain protocols. We did not mention it so far because the specifics of our work did not call for it so far. For any practical application, however, it is a serious issue. Fortunately, there are already efforts to solve this problem, like in [13] and [10]. In future efforts, when a blockchain protocol is created specifically for the purpose of creating a system as proposed in this work, it would need to build on results from [13] or [10] to achieve acceptable scalability.

During development of the system we found ourselves updating contract code frequently, as is common in software development in these times. It is also very common that systems evolve. Since the core code of this system is fixed in smart contracts, evolving this system is highly impractical. It would be possible to move to a completely new network, after consensus on such a decision has been reached, but this brings along a lot of overhead. It would be better to have the the code of a contract variable, which can be updated when a

positive vote for this has a majority. All real decision making would be done off the chain, by representatives of participating parties, with technical expertise. The voting on the chain would only be to make the transition official, perhaps with the new code already staged, ready to replace the old code when the rules for acceptance of new code have been satisfied. This is another reason why a blockchain specifically for the goal of supporting this system is necessary.

6 Conclusion

In this work, we set out to study the feasibility of creating a system to facilitate trade, borne on a blockchain. In our literature research, we found that such system can have high impact on the markets in which they are used, although every practical example we found was from the financial sector. We also found that attempts at similar systems have been made in the past, both with and without the use of blockchain technology. With our insights from the related literature, we first defined a type of bulk commodity market for which we would design our system. With this in mind, we defined a use case on which we based our discussion of requirements. We designed a system, for which we created a minimal proof of concept implementation based on the Ethereum blockchain. Testing an elemental function of the system, its speed seemed to be sufficient. Nonetheless, it appeared from our evaluation that, although such a system would be feasible, the blockchain implementation chosen by us does not have all the necessary qualities expected from it. While other implementations may have what our chosen implementation is missing, all suffer from a lack of scalability. This, however, is expected to be solved in the future. From all this we conclude that it is possible to build a system as we describe that satisfies all the mentioned requirements and is scalable, but that it is highly recommended to build a blockchain implementation specifically for this purpose, while making use of all the available partial solutions. During our efforts to implement our proof of concept, we found that the blockchain implementation we used is still very new and much improvement is needed to better facilitate development of systems based on it. From this, we learned that it is advisable to develop an application based on this blockchain implementation in small increments, and unit test it as well as possible. This way, the least impediment should be experienced from the difficulty in debugging such an application. It should be noted that this work is only going into the specifics of a system that aims to facilitate trade. It does not explore the non-technical effects it may have on a market in which it may be put to use.

References

[1] A. Abidin, A. Aly, S. Cleemput, and M. A. Mustafa, “Secure and privacy-friendly local electricity trading and billing in smart

grid," *CoRR*, vol. abs/1801.08354, 2018. [Online]. Available: http://arxiv.org/abs/1801.08354

[2] E. Androulaki, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, and et al., "Hyperledger fabric," *Proceedings of the Thirteenth EuroSys Conference on - EuroSys '18*, 2018. [Online]. Available: http://dx.doi.org/10.1145/3190508.3190538

[3] B. A. Ayed, "A conceptual secure blockchain-based electronic voting system," *International Journal of Network Security & Its Applications (IJNSA)*, vol. 9, no. 3, May 2017.

[4] J. Benet. (2014) Ipfs - content addressed, versioned, p2p file system. Protocol Labs, Inc. [Online]. Available: https://github.com/ipfs/ipfs/blob/master/papers/ipfs-cap2pfs/ipfs-p2p-file-system.pdf

[5] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "The making of keccak," *Cryptologia*, vol. 38, no. 1, pp. 26–60, 2014.

[6] V. Buterin *et al.* A next-generation smart contract and decentralized application platform. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper

[7] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *CoRR*, vol. abs/1710.09437, 2017. [Online]. Available: http://arxiv.org/abs/1710.09437

[8] L. Cardella, J. Hao, I. Kalcheva, and Y. Ma, "Computerization of the equity, foreign exchange, derivatives, and fixed-income markets," *Financial Review*, vol. 49, no. 2, pp. 231–243, 2014.

[9] A. Daffurn Lewis, "A matchmaking algorithm for a blockchain-based trading platform," B. Sc. thesis, University of Groningen, 2018.

[10] R. Dennis, G. Owenson, and B. Aziz, "A temporal blockchain: A formal analysis," in *International Conference on Collaboration Technologies and Systems*, Oct. 2016, pp. 430–437.

[11] C. Dwork and M. Naor, "Pricing via processing or combatting junkmail," in *Advances in Cryptology — CRYPTO '92*, no. 12, Aug. 16 – 20, 1992.

[12] M. Fan, J. Stallaert, and A. B. Whinston, "The design and development of a financial cybermarket with a bundle trading mechanism," *International Journal of Electronic Commerce*, vol. 4, no. 1, Sep. - Nov. 1999.

[13] X. Feng, J. Ma, Y. Miao, Q. Meng, X. Liu, Q. Jiang, and H. Li, "Pruneable sharding-based blockchain protocol," *Peer-to-Peer Networking and Applications*, 11 2018.

[14] M. A. Goldstein, P. Kumar, and F. C. Graves, "Computerized and high-frequency trading," *The Financial Review*, vol. 49, pp. 177–202, 2014.

[15] G. Ishmaev, "Blockchain technology as in institution of property," *Metaphilosophy*, vol. 48, no. 5, Oct. 2017.

[16] D. Kennedy, "The machine in the market: Computers and the infrastructure of price at the new york stock exchange, 1965–1975," *Social Studies of Science*, vol. 47, no. 6, Dec. 2017.

[17] M. Y. Khan, M. F. Zuhairi, A. T., T. Alghamdi, and J. A. Marmolejo-Saucedo, "An extended access control model for permissioned blockchain frameworks," *Wireless Networks*, Mar 2019.

[18] H. Levecq and B. W. Weber, "Electronic trading systems: Strategic implications of market design choices," *Journal of Organizational Computing and Electronic Commerce*, vol. 12, no. 1, pp. 85–103, 2002.

[19] P. H. Madore. Ethereum classic might have been hit by a 51% attack. [Online]. Available: https://www.ccn.com/ethereum-classic-might-have-been-hit-by-a-51-attack/

[20] P. McCorry, S. Shahandashti, and F. Hao, "A smart contract for boardroom voting with maximum voter privacy," in *Financial Cryptography and Data Security*, Jan. 2017.

[21] M. Mihaylov, S. Jurado, N. Avellana, K. Van Moffaert, I. Magrans de Abril, and A. Nowé, "Nrgcoin: Virtual currency for trading of renewable energy in smart grids," in *International Conference on the European Energy Market (EEM14)*, no. 11, May 28 – 30, 2014.

[22] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[23] B. Notheisen, J. B. Cholewa, and A. P. Shanmugam, "Trading real-world assets on blockchain," *Business & Information Systems Engineering*, vol. 59, no. 6, Dec. 2017.

[24] H. R. Stoll, "High speed equities trading: 1993–2012," *Asia-Pacific Journal of Financial Studies*, vol. 43, pp. 767–797, 2014.

[25] C. A. Waldspurger and W. E. Weihl, "Lottery scheduling: Flexible proportional-share resource management," in *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI '94, 1994.

[26] O. Williams-Grut, "The crypto boom is like the dotcom bubble but that's not a bad thing: 'selling crypto now is like selling apple in 2001'," https://www.businessinsider.nl/ico-dotcom-bubble-yoni-assia-etoro-crypto-blockchain-joseph-lubin- bitcoin-ethereum-2018-6/.

[27] J. Wilmoth. Ethereum classic losses top $1 million after 51% attack: Coinbase research. [Online]. Available: https://www.ccn.com/ethereum-classic-losses-top-1-million-after-51-attack-coinbase-research/

[28] G. Wood. (2019, May) Ethereum: a secure decentralised generalised transaction ledger. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

A Benchmark test results

The test setup was as follows. A JavaScript script was written that would deploy a `Marketplace` contract, publish multiples of 10 offers, from 10 up to and including 100, wait for each offer to be included in the blockchain and then perform a check to see if the offer was properly added. The offers are published one by one, each in a separate transaction. A shell script was then written to run the actual benchmark with different numbers of offers. The test was run with a single trader running on a single machine. The results below were obtained by running the script several times with NodeJS at version 11.2.0 and the Go Ethereum implementation at version 1.8.20. This was done on a Lenovo Ideapad 320 laptop with an Intel® Core™ i5-8250U CPU running at 1.60GHz running Ubuntu Linux at version 16.04. To run the benchmark on your own machine, the instructions in the readme on the repository can be followed[14].

```
will perform benchmark by publishing and retrieving 10 offers
starting miner with 1 thread
10 offers published and retrieved in 2.079 seconds.
----------------------
will perform benchmark by publishing and retrieving 20 offers
starting miner with 1 thread
20 offers published and retrieved in 2.106 seconds.
----------------------
will perform benchmark by publishing and retrieving 30 offers
starting miner with 1 thread
30 offers published and retrieved in 4.055 seconds.
----------------------
will perform benchmark by publishing and retrieving 40 offers
starting miner with 1 thread
40 offers published and retrieved in 6.111 seconds.
----------------------
will perform benchmark by publishing and retrieving 50 offers
starting miner with 1 thread
50 offers published and retrieved in 6.155 seconds.
----------------------
will perform benchmark by publishing and retrieving 60 offers
starting miner with 1 thread
60 offers published and retrieved in 8.195 seconds.
----------------------
will perform benchmark by publishing and retrieving 70 offers
starting miner with 1 thread
70 offers published and retrieved in 390.873 seconds.
----------------------
will perform benchmark by publishing and retrieving 80 offers
starting miner with 1 thread
80 offers published and retrieved in 1749.19 seconds.
----------------------
```

[14] `https://github.com/pieterDekker/blockhain-trading-platform`

```
will perform benchmark by publishing and retrieving 90 offers
starting miner with 1 thread
90 offers published and retrieved in 12.675 seconds.
----------------------
will perform benchmark by publishing and retrieving 100 offers
starting miner with 1 thread
100 offers published and retrieved in 13.696 seconds.
----------------------
```

```
will perform benchmark by publishing and retrieving 10 offers
starting miner with 1 thread
10 offers published and retrieved in 2.069 seconds.
----------------------
will perform benchmark by publishing and retrieving 20 offers
starting miner with 1 thread
20 offers published and retrieved in 2.114 seconds.
----------------------
will perform benchmark by publishing and retrieving 30 offers
starting miner with 1 thread
30 offers published and retrieved in 8.228 seconds.
----------------------
will perform benchmark by publishing and retrieving 40 offers
starting miner with 1 thread
40 offers published and retrieved in 20.389 seconds.
----------------------
will perform benchmark by publishing and retrieving 50 offers
starting miner with 1 thread
50 offers published and retrieved in 10.284 seconds.
----------------------
will perform benchmark by publishing and retrieving 60 offers
starting miner with 1 thread
60 offers published and retrieved in 26.769 seconds.
----------------------
will perform benchmark by publishing and retrieving 70 offers
starting miner with 1 thread
70 offers published and retrieved in 12.879 seconds.
----------------------
will perform benchmark by publishing and retrieving 80 offers
starting miner with 1 thread
80 offers published and retrieved in 26.85 seconds.
----------------------
will perform benchmark by publishing and retrieving 90 offers
starting miner with 1 thread
90 offers published and retrieved in 203.244 seconds.
----------------------
will perform benchmark by publishing and retrieving 100 offers
starting miner with 1 thread
100 offers published and retrieved in 24.741 seconds.
----------------------
```

```
will perform benchmark by publishing and retrieving 10 offers
starting miner with 1 thread
10 offers published and retrieved in 2.091 seconds.
----------------------
will perform benchmark by publishing and retrieving 20 offers
starting miner with 1 thread
20 offers published and retrieved in 4.119 seconds.
----------------------
will perform benchmark by publishing and retrieving 30 offers
starting miner with 1 thread
30 offers published and retrieved in 4.137 seconds.
----------------------
will perform benchmark by publishing and retrieving 40 offers
starting miner with 1 thread
40 offers published and retrieved in 16.327 seconds.
----------------------
```

```
will perform benchmark by publishing and retrieving 50 offers
starting miner with 1 thread
50 offers published and retrieved in 12.291 seconds.
----------------------
will perform benchmark by publishing and retrieving 60 offers
starting miner with 1 thread
60 offers published and retrieved in 67.066 seconds.
----------------------
will perform benchmark by publishing and retrieving 70 offers
starting miner with 1 thread
70 offers published and retrieved in 10.388 seconds.
----------------------
will perform benchmark by publishing and retrieving 80 offers
starting miner with 1 thread
80 offers published and retrieved in 21.123 seconds.
----------------------
will perform benchmark by publishing and retrieving 90 offers
starting miner with 1 thread
90 offers published and retrieved in 134.012 seconds.
----------------------
will perform benchmark by publishing and retrieving 100 offers
starting miner with 1 thread
100 offers published and retrieved in 59.117 seconds.
----------------------
```

B Alterations to Geth

Geth is the Go implementation of Ethereum. We used Geth completely unaltered, except for one line of code. The reason behind that is explained here.

In Ethereum, block difficulty increases when blocks are mined too fast, and block difficulty decreases when they are mined too slow. This is to keep the time to mine a block at a certain target[15], while this target increases gradually over time [16]. For development purposes, we changed the line of code that was responsible for recalculating the difficulty so that the difficulty was constant.

```
1 // CalcDifficulty is the difficulty adjustment algorithm. It
      ↪ returns
2 // the difficulty that a new block should have when created at time
3 // given the parent block's time and difficulty.
4 func (ethash *Ethash) CalcDifficulty(chain consensus.ChainReader,
      ↪ time uint64, parent *types.Header) *big.Int {
5     return CalcDifficulty(chain.Config(), time, parent)
6 }
```

Listing 1: The original code, as of version 1.8.20 [17]

```
1 // CalcDifficulty is the difficulty adjustment algorithm. It
      ↪ returns
2 // the difficulty that a new block should have when created at time
3 // given the parent block's time and difficulty.
4 func (ethash *Ethash) CalcDifficulty(chain consensus.ChainReader,
      ↪ time uint64, parent *types.Header) *big.Int {
```

[15] As mentioned in the rationale section in https://github.com/ethereum/EIPs/blob/master/EIPS/eip-100.md

[16] As explained in https://github.com/ethereum/EIPs/blob/master/EIPS/eip-649.md

[17] https://github.com/ethereum/go-ethereum/releases/tag/v1.8.20

```
5       return  big.NewInt(1)
6   }
```

Listing 2: The modified code