Automating Bulk Commodity Trading Using Smart Contracts

Dekker, Pieter; Andrikopoulos, Vasilios

Published in:
2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)

DOI:
10.1109/DAPPS49028.2020.00006

Automating Bulk Commodity Trading Using Smart Contracts

Pieter Dekker and Vasilios Andrikopoulos
Computer Science Department
University of Groningen
Groningen, the Netherlands
Email: p.dekker.2@student.rug.nl, v.andrikopoulos@rug.nl

***Abstract*—Blockchain technology as an approach for decentralized and distributed ledgers provides a natural fit solution for trading applications. A number of existing approaches have been developed in the recent years aiming to support trading in various markets by means of this technology. However, the role of blockchain in these approaches is in many cases confined to recording the consensus on the transferred assets prescribed by trade agreements that are made outside of the chain itself. The premise of this work is that blockchain smart contracts can actually be leveraged to automate the creation of those trading agreements, adding a new level of autonomy in the operation of the market itself. For these purposes we design and realize as a proof of concept an Ethereum-based network of peers trading electricity automatically using Solidity smart contracts. Beyond introducing automated matchmaking capabilities for bids in the market, the proposed approach also offers balancing of computational effort across peers. An additional benefit is that it minimizes the influence of single nodes to the operation of the automated market by moving into a proof of stake-like scheme of rotating leadership for block mining. Experimental evaluation of the work shows promise, but further work is required for a full blown solution. Lessons learned for Ethereum-based solution development are also offered.**

***Index Terms*—blockchain, smart contracts, energy trading, buyer-seller matching, Ethereum**

I. Introduction

Trading is an activity that can be described as a two-sided agreement to transfer assets between parties. 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 [1]. Others, like high speed trading are a prime example of how such developments can bring about radical changes [2], [3]. Blockchain technology, as a decentralized, distributed ledger in which the content of the ledger is divided into blocks linked through cryptographic hashing and stored by all participants in a network as introduced in [4], is designed for tracking the transfer of digital assets. In light of the ongoing efforts for automation and digitization, it becomes of critical interest to explore this apparent compatibility between trading and blockchain.

There are a number of approaches in the literature of recent years that have explored this space. For example, Notheisen et al. [5] develop a system for cars trading; Chiu and Koeppl discuss using blockcahin for settlement in asset trading [6]; even data trading [7] can be supported by blockchain. The main theme and focus on these approaches, however, is on providing the means for digitally recording the transfer of assets in an immutable and auditable manner by using blockchain technology [8]. Traders (either humans or systems/algorithms) are to make decisions on the proposed trades, and blockchain is to be used to reflect the outcomes of these trades as agreements. Smart contracts are used to represent these binding agreements, and to enforce the obligations of the participating entities [9].

In this way of operating, however, we see a missed opportunity to take a step further, and actually automate, where possible, the trade itself. Smart contracts, in this case, are to be used not only for enforcing the trade agreements, but also as the means for producing these agreements in the first place. This work investigates the feasibility of realizing a system of automated trading over a blockchain network, using smart contracts for this purpose. Automated trading is not a novel concept per se. Fan et al. [10], for example, 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 these bundles by matching buy orders in one bundle with sell orders in another bundle and so on, 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 out after a matching run. These assets are returned to the trader after a round. The approach we propose in this work is similar in spirit to this mode of operation but without the need for an exchange to do the matching.

In order to scope down the task at hand, we focus on *bulk commodities* trading, i.e. assets[1] that are traded in large quantities such as hydrocarbon products, grain, coal etc., and pick the *energy trading* domain, where energy is traded between peers (producers and consumers of electricity) as our use case. Energy trading systems appear to be great candidates for adoption of the blockchain technology since it

[1]Technically speaking, assets are typically either *break bulk* commodities like cars or electronic devices, or *bulk* commodities available in arbitrary volumes, like energy or grain.

978-1-7281-6978-1/20/$31.00 ©2020 IEEE
DOI 10.1109/DAPPS49028.2020.00006

offers many advantages in comparison to traditional systems in the domain [12], [13]. The main research question that this work is aiming to answer can therefore be summarized as: *to what extent can blockchain-supported trading of single types of bulk commodities, and specifically energy, be automated by means of smart contracts?* Similarly to what Fan et al. propose in [10], our system will perform matching on a set of representations of the traders intention to buy or sell a commodity, that is an amount of electricity in bulk, as the means for automating the trading between interested parties. In this respect, it focuses on the *buyer-seller matching* aspect of transaction handling in the classification of related works by Wang et al. [11].

The rest of this paper is structured as follows. The domain of the use case is analyzed in Section II, and requirements on a desired solution are consequently elicited. Our proposed system design is discussed in Section III; the proof of concept implementation of our proposal and the evaluation of our solution are elaborated in Section IV. Section V presents related work from blockchain-supported trading approaches, with an emphasis on the identified use case, and positions our approach with respect to the state of the art. Section VI closes this paper with a summary of the main points and future work.

II. System Requirements

A. Use Case

For the purposes of this work, we consider a domain consisting of a set of markets with specific characteristics. More specifically, as discussed in the previous section, we are looking at markets that involve a single bulk commodity. Examples of such markets 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 the second characteristic: within such markets, an intent to buy or sell is only temporarily relevant. After an arbitrary but short amount of time, these intentions become obsolete if not materialized. Take the example of a wind park during a week of strong winds. The owner of the park would want to sell electricity before generating it, 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 consider high-speed trading as out of the scope of this work. We therefore 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. In addition, 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, acting potentially as both producers and consumers (*prosumers*). In short, this work is aimed 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*. The energy trading market acts as the specific instantiation of the domain under consideration for the purposes of this paper, but the provided solution (and the discussion which follows) seeks to be equally applicable to similar markets in the same domain.

The users of a system in this domain, 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, the point in time where the amount is or should be available, and at what price per unit. The unit used is assumed to be system-wide and is therefore left out in the details of the 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, or at least in good faith, they can be 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.

B. Requirements

Based on this use case description above, the following requirements on a blockchain-based solution are elicited:

- *R1* 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 in which time window. Offers and demands that are compatible are matched automatically. Resulting *matches* must be confirmed by both parties involved in a match in order to take effect.
- *R2* The offering party has to be able to make a claim that it sent a certain volume of the commodity, and the demanding party must be able to confirm such a claim, formulated as a *trade agreement*. When the agreed upon volume is confirmed by both parties, a *payment agreement* must be created automatically to reflect this fact based on the contents of the trade agreement.
- *R3* The demanding party must make a claim that it sent an amount of payment and the offering party to confirm it, as per the generated payment agreement. When the *transfer of funds* is confirmed, the interaction between the two parties is finished and none of the agreements mentioned above can be altered after the fact.

R4 All the actions that facilitate the functionality described above should be *auditable and reproducible*, so that any dispute over a trade interaction concerning one or more functionalities that the system offers can be settled by reproducing one or more steps in the interaction.

R5 The *ownership and operating burden* of the system should be distributed *fairly* among participants. This also includes the amount of data to be stored per participant in the system. The required computational and storage prerequisites should not be so restrictive that it is impossible for traders to begin participation in the network later in its lifetime.

R6 The system should perform *fast enough* for an action to be effective within an acceptable amount of time (arbitrarily set to an hour for the use case), subject to a possible counter party's latency of response.

To the above, we also need some technical requirements on the blockchain solution to be created for the use case, affecting the selection of an appropriate technology stack to implement the foreseen solution. More specifically:

T1 The blockchain must allow the maintenance of stateful accounts; alterations to the state of the accounts is performed by means of transactions.

T2 The blockchain must store all changes to the states of accounts as part of its transaction history.

T3 It must be possible to control which participant in the network creates a block, and how many transactions are stored per block, while still respecting *R5*. This constraint is to limit the network to the participants in a market.

III. Design

A. Technologies

As a first step in our design process, we decide on the blockchain platform to be adopted for further developing our solution. Since there are a number of available blockchain platforms in the field with an acceptable perceived level of maturity, we opt for adopting one of them instead of developing from scratch. Looking at the technical requirements *T1–T3*, we choose the Ethereum blockchain, described in [14]. The Ethereum blockchain is an account-based blockchain, where accounts send transactions amongst each other, satisfying *T1*. New accounts are created by sending transactions to a zero-account, which is by definition not-owned. There are two types of accounts, *Externally Owned Accounts (EOAs)* and *Contract Accounts* (or simply contracts). EOAs are controlled by private keys and do not contain code, but are used to send transactions to other accounts. Contracts consist of the state of account, including its balance, and some contract code to be invoked as part of the execution of a transaction. All transactions are stored on the chain, and any previous state of an account can be reproduced (*T2*). Despite the fact that Ethereum is meant for public blockchain networks, with the way contracts can be written for the Ethereum blockchain, *T3* can also be satisfied by the specifics of the design. The Ethereum 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. A detailed technical description of the Ethereum blockchain can be found in [15].

For our design we aim to leverage the use of smart contracts as the means of realizing the automated trading mechanisms. There are a number of available smart contract languages that would fit our choice of blockchain platform, for example Serpent[2], Vyper[3], LLL[4], and Solidity[5]. Since Solidity actually comes recommended from the Ethereum developers, is still actively developed, and is of sufficient abstraction level to facilitate development we opt for it. 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 compiled to Ethereum Virtual Machine (EVM) bytecode and executed by the EVM, resulting into the state of accounts being altered, or transactions being sent with the compiled code as payload to be executed.

In order to keep the amount of data on the chain to a minimum (*R5*) and considering also the requirements on performance (*R6*), we supplement the Ethereum blockchain with the Inter Planetary File System (IPFS) [16], allowing us to off-load any data not necessary for the chain to it. IPFS is a distributed file storage system with an easy to use interface. It is based on a collection of protocols inspired by technologies such as Distributed Hashtables, BitTorrent, SPS and Git to provide a low-latency distributed file system.

In the following we provide a high-level view of the architecture of the proposed system and its operating principle, before delving into more depth on aspects of particular interest.

B. High-level Architecture

The proposed system consists of two groups of smart contracts, as summarized by Fig. 1: Marketplace, Matches, TradeAgreements and PaymentAgreements, that provide the functionality prescribed by *R1–R3*, and Traders and Leader which are used to satisfy *R5–R6*. More specifically, transactions to publish offers and demands are connected to the Marketplace contract. Marketplace contains two collections, one for offers and one for demands, corresponding to respective IPFS file paths. These files on the IPFS contain all the data related to the offers and demands, 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. Offers and demands are matched automatically, and the results of this matching are listed in the Matches contract. When the Matches contract has received signed confirmation messages[6] from

[2]https://github.com/ethereum/serpent

[3]https://github.com/ethereum/vyper

[4]https://solidity.readthedocs.io/en/v0.5.12/lll.html (deprecated)

[5]https://github.com/ethereum/solidity

[6]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".

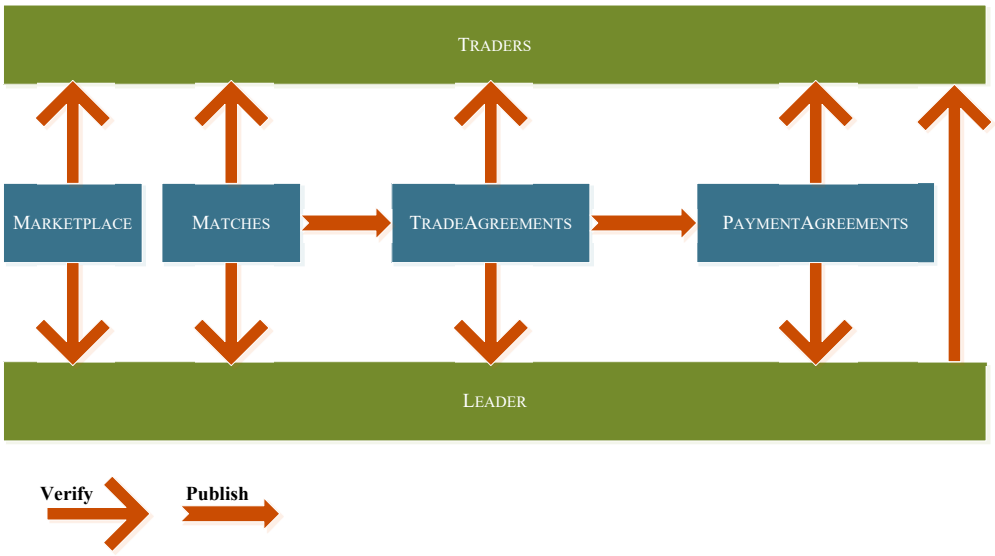

Fig. 1. The smart contracts comprising the proposed system, and their interactions

both parties involved, it lists the match as an agreement in the TRADEAGREEMENTS contract defining the terms of the exchange to take place.

The offering party, after sending a certain volume, claims that the actual volume sent has changed; if the demanding party, after receiving said volume, confirms this claim, the actual volume sent for the agreement in question is updated in TRADEAGREEMENTS. If the actual volume sent matches the amount of the total volume to be sent, the delivery is complete, and TRADEAGREEMENTS creates an entry in the PAYMENTAGREEMENTS contract. 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. When the amount is confirmed to be paid, the trade interaction is finished and no more updates can be done. The entire interaction is now auditable through the transactions that led to the current state of accounts. Disputes in terms of non-compliant with TRADEAGREEMENTS volumes and funds transfers not matching the PAYMENTAGREEMENTS are reflected on the chain itself, and can be used as evidence for further adjudication and/or conflict resolution (outside of the scope of this work).

As shown in Fig. 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 (requirement *T3*), identified by their EOA address. The last contract in the figure, LEADER, is used to enforce the fair distribution of computational effort required for the automated matchmaking (*R5*) across all network members. These two issues (matchmaking and leadership) are discussed in more detail in the following.

C. Automated Matchmaking

For matchmaking purposes we use an adaptation of the algorithm proposed in [17], itself an adaptation of previous work [18], [19] on automated contract generation for Web Services. More specifically, the algorithm that Daffurn Lewis proposes in [17] for the purpose of matchmaking in a system of the type we are considering can be summarized as follows:

```
1:  function MATCHMAKING(Offers O, Demands D)
2:      Matches M ← ∅
3:      for all o ∈ O do
4:          for all d ∈ D do
5:              mQ ← MATCHQUANTITY(o, d)
6:              mP ← MATCHPRICE(o, d)
7:              mT ← MATCHTIMEWINDOW(o, d)
8:              if mQ > 0 ∧ mP > 0 ∧ mt > 0 then
9:                  M ← M ∪ {
10:                 offerAddress ← o.address,
11:                 demandAddress ← d.address,
12:                 quantity ← mQ,
13:                 price ← mP
14:                 timeWindow ← mT}
15:                 D ← D − d
16:                 O ← O − o
17:             end if
18:         end for
19:     end for
20:     return M
21: end function
```

The algorithm performs matchmaking on two lists, one list of offers and one list of demands (called bids in [17]). It iterates over the list of offers and finds the first match in the

list of demands. Matching is based on treating the prices and quantities in the demands and offers as intervals, allowing the application of Allen's interval algebra [20] to examine their relations, in the same manner as discussed in [19]. If the offered quantity in an offer is sufficient to satisfy demand (i.e. function MATCHQUANTITY returns non-zero value) for a price that is at most equal to the desired one (MATCHPRICE> 0) in the prescribed time window (MATCHTIMEWINDOW> 0), then a match is created by combining the two and removing them from their respective lists for further matching. The algorithm iterates over all offers until the list is empty or exhausted.

As a result, 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. While a more sophisticated matching scheme can be added to our system, this first-fit basis one is sufficient to satisfy the requirements we identified in the previous section. What's more, due to its simplicity, the algorithm is also very fast: in the experiment discussed in [17], it takes only 1 second to match 17500 randomly generated offers and demands when used as a NodeJS package running on an older (circa 2014) generation laptop. This means that the algorithm is more than fast enough to satisfy *R6*.

Nevertheless, we also have to consider that if the algorithm itself is implemented as a smart contract, it will have to be run by every participant any time matchmaking is required, which is less than desirable. Therefore the algorithm is run off the chain by a *selected leader* defined in the LEADER contract, in *rounds* (see below). The input used and the output produced is recorded in a file on the IPFS which is referred to explicitly by the MATCHES contract. This way, matchmaking results can always be verified by anyone in the network, if necessary.

D. Rounds & Leadership

The introduction of rounds and leadership is required in our system since for the size of networks prescribed by the adopted use case, i.e. at maximum in the order of thousands nodes, the proof of work algorithm used by the Ethereum blockchain for consensus building (similar to the one by Dwork and Naor [21]) can be easily subverted by a peer with disproportionate amount of hardware. For this purpose, we propose that the recording of mutations to the system, as effected by transactions, is organized in rounds. The activities in each round are summarized in Fig. 2. The proposed scheme is somewhat similar to the proof-of-stake consensus mechanism to be introduced in Ethereum 2.0[7], codename Serenity. However, at the point of writing this paper, this release is still in the future; we therefore base our work as we will discuss further in Section IV on a version of Ethereum still relying on proof-of-work, with our leadership mechanism implemented on top of it.

More specifically: in each round, one of the traders acts as the *round leader*. At the start of each round (top left of Fig. 2), the leader notifies the other traders that they can begin sending bids (that is, demands or offers) as transactions to the MARKETPLACE contract. At the end of the round (bottom left of Fig. 2), and before the matchmaking starts, the leader notifies the other traders that the window for sending bids in the current round is closed. The leader then retrieves the offers and demands from the MARKETPLACE contract and runs the matchmaking algorithm as discussed in the previous section. When the matchmaking is done, the leader records the result of the round in one or more blocks on the chain, listing the offers and demands included in matchmaking, and the matches made by the algorithm, in a file on the IPFS, and storing the path to this file in the MATCHES contract to allow future auditing. Matches are required to be digitally signed by both trading parties (producer and consumer) in order to be added to the TRADEAGREEMENTS contract, and eventually to PAYMENTAGREEMENTS, as discussed in Section III-B. Since this part of the operation of the proposed system is outside of the scope of this paper, we are omitting it from Fig. 2 for clarity and simplicity.

Unmatched offers and demands, and matches from previous rounds that are not accepted by both parties are discarded at the end of the round. Each offer and demand is therefore valid for only one round. This means that the responsibility of keeping offers and demands available until they are matched or obsolete lies with the traders themselves. This is an acceptable situation in light of the benefit of the greater scalability of the chain it provides. Newly made matches are published as the final action of the leader in one or more blocks, before unseating themselves and starting the leader selection process (right hand side of Fig. 2).

Leader selection is based on a lottery scheduling scheme similar to [22] as the means of enforcing the fair distribution of work throughout the network (*R5*). At the end of each round, the currently selected leader, trader T_a, draws a lot from the collection of all issued tickets. The owner of the ticket, say T_b, is selected as the leader for the next round, and the ticket is marked as available again, taking it away from T_b. If T_b now holds no tickets, they are issued an amount of new tickets proportionate to the standard deviations σ_b of the sum of transactions R_i in all the rounds L that T_b was leader $\sum_{i \in L_{T_b}} R_i$ from the mean transactions per round so far $\bar{R}$ (minimum one ticket). The amount of tickets issued is limited by a maximum upper value, which is to be determined experimentally for the purpose of preventing unequal chances at becoming a leader between traders. Incorporating this mechanism into the proposed solution is outside of the scope of this work, and the matter for future work. Leadership is enforced in practice by requiring all transactions in a block to have as block creator's address the leader of this round (as defined in the LEADER contract). This way, only transactions included in a block by the current leader can be valid and lead to valid blocks; therefore only the leader can create blocks.

This scheme ensures that each leader performs an equally distributed amount of work and therefore no trader has a

[7]https://docs.ethhub.io/ethereum-roadmap/ethereum-2.0/proof-of-stake/

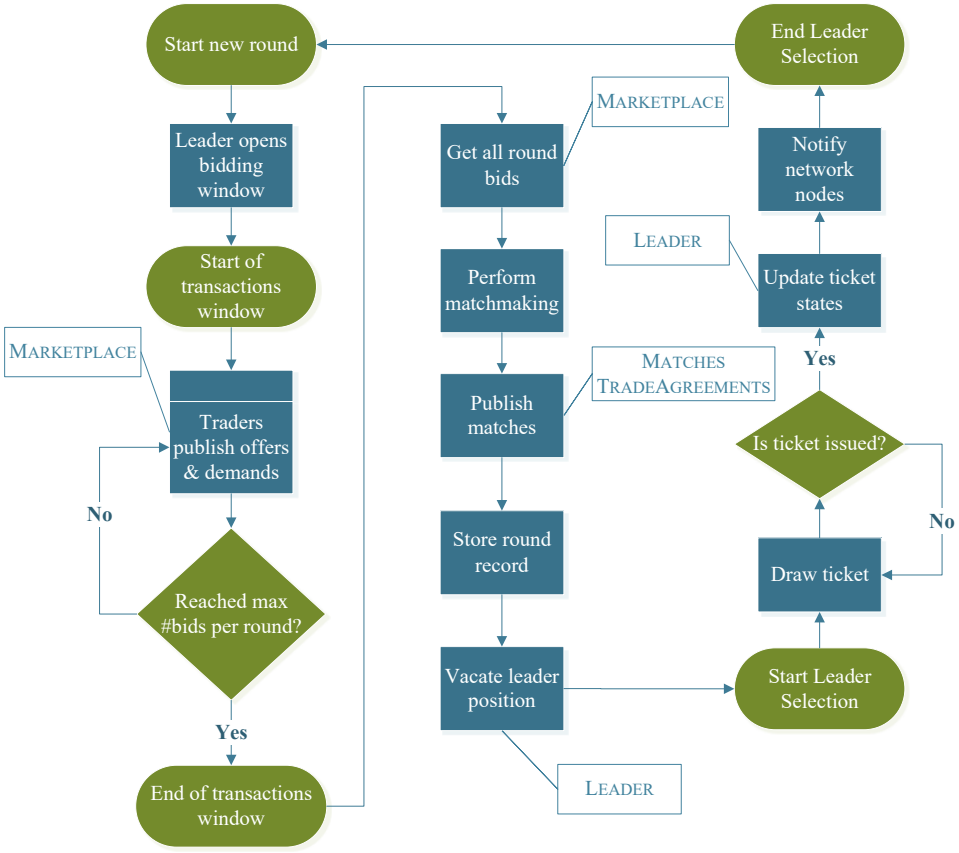

Fig. 2. The flow of information in a system round including the leader selection process, and the affected contracts in the system

disproportionate stake in the system. Furthermore, new traders joining an existing network are eligible for the maximum between $\frac{1}{\sigma}\sum_{i\in L_{T_{new}}} R_i$ and the maximum number of tickets that can be held by a trader at a time. This allows T_{new} to catch up with the other traders in the network without giving the trader an unnecessary amount of control to the network. Since we ensure to ensure reproducibility and auditability (*R4*) also for the ticket draw itself, we use Ethereum's built-in hash function originally introduced in [23] as a pseudo-random generator. The number and time stamp of the block in which the leader selection occurs are given as input to the function.

One issue that we did not touch in the discussion above is the actual amount of work for the leader per round. While the block size B is fixed, the amount of transactions to be included per block, and the amount of blocks to be generated per round still needs to be defined. In order to deal with this issue, we make the assumption that the amount of work to be executed per transaction (matching) is the same for every transaction, considering also transactions that are neither offers nor demands — for example the confirmations of matches and the creation of trade agreements.

We start with the observation that the maximum number of matches $\max|M|$ that can be made in a round is the minimum between the number of offers $|O|$ and the number of demands $|D|$ in that round. The total number of transactions in the i-th round R_i is therefore:

$$R_i = |O_i| + |D_i| + \min(|O_i|, |D_i|) + \rho_i$$

where ρ_i is the number of other transactions to be processed. Let R_{max} be the experimentally defined maximum amount of transactions per round as discussed above. The leader has to stop including transactions from other traders in blocks as soon as it holds that

$$R_{max} - \min(|O_i|, |D_i|) > |O_i| + |D_i| + \rho_i$$

Assuming that τ_i transactions are included in R_i, then it follows that at most

$$\lceil \frac{\tau_i + min(|O_i|, |D_i|)}{B} \rceil$$

blocks will be created by the leader in this round. In this manner, as the amount of offers and demands changes with the amount of participants to the network over time, the amount of work required by each round leader will fluctuate accordingly in response (*R5*).

IV. Implementation & Evaluation

A. Implementation

In order to demonstrate the feasibility of our proposal we implemented the proposed system in a proof of concept trading platform[8]. In terms of adopted software, we used the Go implementation of the Ethereum system at version 1.8.20[9]. The system followed the Byzantium hard fork of the Ethereum protocol[10]. Solidity release 0.4.25[11] was used for the smart contracts. We also used the Javascript implementation of IPFS at version 0.35[12].

```
function publishOffer(bytes _ipfs_file_id) public {
  if (current_offers_amount == _offers.length) {
    _offers.push(_ipfs_file_id);
  } else {
  _offers[current_offers_amount] = _ipfs_file_id;
  }
  emit NewBid(current_offers_amount);
  current_offers_amount++;
}

function publishDemand(bytes _ipfs_file_id) public {
  if (current_demands_amount == _demands.length) {
    _demands.push(_ipfs_file_id);
  } else {
  _demands[current_demands_amount] = _ipfs_file_id;
  }
  emit NewBid(current_demands_amount);
  current_demands_amount++;
}
```

Listing 1. Solidity code for publishing an offer or a demand on the chain

Listing 1 is a snippet from the MARKETPLACE contract showing the functions handling the publishing of offers and demands to the chain by means of their IPFS handles.

One issue with the adoption of Ethereum as the implementation network is that of controlling participation to it. Ethereum, by default, is a network in which anyone can begin participating in by simply knowing its identification number, discovering other participants and adhering to the protocols. This kind of open protocol is undesirable for our system. 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 Hyperledger Fabric [24]. We decided that this issue lies outside of the scope of our proof of concept implementation and approaches like [25] can be used to resolve this issue in practice.

An additional issue that the adoption of Ethereum raises is that of mining difficulty. In the proposed rotating leadership scheme presented in the previous section, the validity of a block is dependent only on the identity of the creator, that is, the selected leader for the round. A digital signature of the block creator is sufficient for these purposes, which is by default included in all valid Ethereum transactions. It is therefore sufficient to configure the network for the lowest possible mining difficulty, and to implement the leadership in rounds through the MARKETPLACE and LEADER smart contracts. For the purposes of this proof of concept, we also opted not to implement the TRADEAGREEMENTS contract, to be updated by the MATCHES contract as shown in Fig. 2. Any trade agreement that is the result of intentional actions of the involved parties has the IPFS file handle of an offer and of a demand with their respective signatures. 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. Implementing these contracts is part of future work.

[8]https://github.com/pieterDekker/blockhain-trading-platform
[9]https://github.com/ethereum/go-ethereum/releases/tag/v1.8.20
[10]https://github.com/ethereum/EIPs/blob/master/EIPS/eip-609.md
[11]https://github.com/ethereum/solidity/releases/tag/v0.4.25
[12]https://github.com/ipfs/js-ipfs/releases/tag/v0.35.0

B. Evaluation

As discussed above, the Ethereum network realization of our proposal is meant as a proof of concept, allowing us to evaluate to what extent the main research question of this work is actually answered by it. Looking at the requirements on a solution identified in Section II-B, it is fairly straightforward to show that our proposed system satisfies most of them. More specifically, by combining Ethereum with IPFS *T1* (stateful accounts) and *T2* (history of state modifications) are met out of the box. Controlled network membership (*T3*) is not straightforward and has been shifted to future work in terms of implementation, but in principle it would be relatively simple to enforce by means of the TRADERS contract, as discussed above. In purely functional terms, the TRADERS, MARKETPLACE, MATCHES, TRADEAGREEMENTS, and PAYMENTAGREEMENTS contracts realize all the functionality required for *R2–R3*. The project repository contains test scripts that allow for confirming this independently.

Applying the lottery-based rotating leadership scheme means that *R5* is also satisfied by definition. An experimental demonstration of the equal burden over time would require the deployment of a large scale network of nodes, which is currently outside of our capabilities, but a target for future work through the adoption of our proposal in existing networks. Auditability and reproducibility of actions is also guaranteed for all phases of each leadership round, down to the drawing of lots for the leadership election (*R4*).

The one requirement that we therefore need to investigate further is that of acceptable performance (*R6*). For this purpose we created a small benchmark, also included in the project repository. The benchmark consists of a script deploying a MARKETPLACE contract, populating it with randomly generated offers to be published in sequential order as separate transactions, and measuring the time it takes to retrieve the transactions from the chain. A shell script which iterates over multiples of 10 from 10 to 100 offers invokes the marketplace simulation script. Running the benchmark on a Lenovo Ideapad 320 laptop with an Intel i5-8250 CPU at 1.6 GHz with 8 GB of RAM running Ubuntu Linux LTE 16.04, shows a turnaround time of at most 1749.2 seconds. Let the average la-

tency between any two traders in the network be 0.3 seconds[13], the number of traders 1000 and let the network topology be entirely linear (i.e. the worst possible case). 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 $999 \times 0.3 + 1749.2 + 999 \times 0.3 = 2348.6$ seconds, a little over 39 minutes, for an offer to be published and the publishing trader to see that in effect. This is well within the limit acceptable by the domain as discussed in Section II. We did not measure the matching latency, but as reported in [17], it takes less than a second to generate all matches and update the MATCHES contract. We can therefore conclude that the performance of the network, at least in this artificial environment, is more than acceptable.

C. Lessons Learned

Ethereum and its contract language Solidity are technologies still very much under development. This means that working with them creates additional difficulties, 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 developing our proof of concept prototype, there was an assertion statement for which the failure case would compile to an invalid opcode (an instruction to the EVM)[14]. 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 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 created confusion, since it seemed something had gone wrong with the transaction itself, while it was simply the assertion that was failing. At the same time, 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 Ethereum project, specifically what exactly has been implemented fully and what is still incomplete, before adopting it in their approach.

V. RELATED WORK

There are a number of efforts towards developing blockchain-based energy trading platforms. A well-known example is NRG-X-Change, introduced in Mihaylov et al. [26]. 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. Other similar industrial approaches are SolarCoin[15], Sunchain[16], the Pylon network[17], and the Power Ledger[18] platform. The interested reader is referred to the survey of the State of the Art in blockchain-based energy trading by Wang et al. [11] for an up-to-date perspective on approaches in this domain.

As discussed in the introductory section, our work aims to automate the buyer–seller matching in these kind of markets. As per [11], matching approaches either use an instantiation of the stable matching problem on bipartite graphs based on node preferences [27] usually aimed at preserving the privacy of participants [28], or an auction-enabling mechanism as in for example [29], [30]. Of particular mention is the Power Ledger platform which provides its own matching algorithms for dealing with demand and supply based on the availability of renewable energy resources in the geographical area of prosumers[19]. With respect to these approaches, the innovation in our proposal is two-fold: first the adoption of an existing efficient mechanism for contract matching which allows us to satisfy the performance requirements on a developed network safely; and second, the introduction of a mechanism which ensures fairness in the distribution of effort across network nodes for the actual matching.

VI. CONCLUSION

Supporting the trading of assets by means of transparently and immutably recording the involved business transactions is an ideal case of the adoption of blockchain technology. Going beyond this record-keeping functionality, in this work we propose to introduce also a degree of automation in the trading of single type bulk commodities like energy by means of automated matchmaking of offers and demands. The system we propose is developed on top of the Ethereum blockchain platform, and aims to address requirements coming from the energy trading domain. A special interest in our system is in balancing the effort required for applying the decentralized matchmaking approach that we introduce. For this purpose we move into a proof-of-stake-like mechanism for block mining done in rounds by rotating leaders selected through lottery drawing. The evaluation of the performance of our developed prototype in a small scale experimental setting shows that our proposal is able to scale as expected.

Future work aims at addressing the shortcomings of the prototype identified in the previous sections. More specifically, the TRADEAGREEMENTS and PAYMENTAGREEMENTS contracts are to be implemented and incorporated in the network. A large scale evaluation of the proposed solution, and a systematic comparison with similar approaches discussed in the related work sections is also a target for the immediate future.

[13] https://wondernetwork.com/pings

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

[15] http://www.solarcoin.org/

[16] http://www.sunchain.fr/

[17] https://pylon-network.org/

[18] https://www.powerledger.io/

[19] https://www.powerledger.io/our-technology/

REFERENCES

[1] D. Kennedy, "The machine in the market: Computers and the infrastructure of price at the new york stock exchange, 19651975," *Social Studies of Science*, vol. 47, no. 6, Dec. 2017.

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

[3] H. R. Stoll, "High speed equities trading: 19932012," *Asia-Pacific Journal of Financial Studies*, vol. 43, pp. 767–797, 2014.

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

[5] 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.

[6] J. Chiu and T. V. Koeppl, "Blockchain-based settlement for asset trading," *The Review of Financial Studies*, vol. 32, no. 5, pp. 1716–1753, 2019.

[7] S. Delgado-Segura, C. Pérez-Solà, G. Navarro-Arribas, and J. Herrera-Joancomartí, "A fair protocol for data trading based on bitcoin transactions," *Future Generation Computer Systems*, 2017.

[8] H. Watanabe, S. Fujimura, A. Nakadaira, Y. Miyazaki, A. Akutsu, and J. J. Kishigami, "Blockchain contract: A complete consensus using blockchain," in *2015 IEEE 4th global conference on consumer electronics (GCCE)*. IEEE, 2015, pp. 577–578.

[9] D. Macrinici, C. Cartofeanu, and S. Gao, "Smart contract applications within blockchain technology: A systematic mapping study," *Telematics and Informatics*, vol. 35, no. 8, pp. 2337–2354, 2018.

[10] 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.

[11] N. Wang, X. Zhou, X. Lu, Z. Guan, L. Wu, X. Du, and M. Guizani, "When energy trading meets blockchain in electrical power system: The state of the art," *Applied Sciences*, vol. 9, no. 8, p. 1561, 2019.

[12] I. Kounelis, G. Steri, R. Giuliani, D. Geneiatakis, R. Neisse, and I. Nai-Fovino, "Fostering consumers' energy market through smart contracts," in *2017 International Conference in Energy and Sustainability in Small Developing Economies (ES2DE)*. IEEE, 2017, pp. 1–6.

[13] E. Mengelkamp, B. Notheisen, C. Beer, D. Dauer, and C. Weinhardt, "A blockchain-based smart grid: towards sustainable local energy markets," *Computer Science-Research and Development*, vol. 33, no. 1-2, pp. 207–214, 2018.

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

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

[16] 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

[17] A. Daffurn Lewis, "A matchmaking algorithm for a blockchain-based trading platform," B. Sc. thesis, University of Groningen, 2018. [Online]. Available: http://fse.studenttheses.ub.rug.nl/18308/1/AlexDaffurnLewis_Thesis.pdf

[18] V. Andrikopoulos, S. Benbernou, and M. P. Papazoglou, "Managing the evolution of service specifications," in *International Conference on Advanced Information Systems Engineering*. Springer, 2008, pp. 359–374.

[19] V. Andrikopoulos, M. Fugini, M. P. Papazoglou, M. Parkin, B. Pernici, and S. H. Siadat, "Qos contract formation and evolution," in *International Conference on Electronic Commerce and Web Technologies*. Springer, 2010, pp. 119–130.

[20] J. F. Allen, "Maintaining knowledge about temporal intervals," in *Readings in qualitative reasoning about physical systems*. Elsevier, 1990, pp. 361–372.

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

[22] 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.

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

[24] 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

[25] 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.

[26] M. Mihaylov, S. Jurado, N. Avellana, K. Van Moffaert, I. Magrans de Abril, and A. Now, "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.

[27] F. Yucel, E. Bulut, and K. Akkaya, "Privacy preserving distributed stable matching of electric vehicles and charge suppliers," in *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*. IEEE, 2018, pp. 1–6.

[28] A. Laszka, S. Eisele, A. Dubey, G. Karsai, and K. Kvaternik, "Transax: A blockchain-based decentralized forward-trading energy exchanged for transactive microgrids," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2018, pp. 918–927.

[29] J. Kang, R. Yu, X. Huang, S. Maharjan, Y. Zhang, and E. Hossain, "Enabling localized peer-to-peer electricity trading among plug-in hybrid electric vehicles using consortium blockchains," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 6, pp. 3154–3164, 2017.

[30] J. Guerrero, A. C. Chapman, and G. Verbič, "Decentralized p2p energy trading under network constraints in a low-voltage network," *IEEE Transactions on Smart Grid*, 2018.