FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Lehrstuhl für Informatik 7

Rechnernetze und Kommunikationssysteme

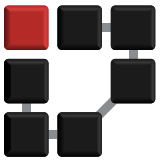
Matthias Seidl

Implementation of Blockchain-Based and Grid-Friendly Local Energy Markets

Masterarbeit im Fach Maschinenbau

16. Oktober 2018

Please cite as:
Matthias Seidl, “Implementation of Blockchain-Based and Grid-Friendly Local Energy Markets,” Master Thesis (Masterarbeit), University of Erlangen, Dept. of Computer Science, October 2018.



Friedrich-Alexander-Universität Erlangen-Nürnberg
Department Informatik
Rechnernetze und Kommunikationssysteme

Martensstr. 3 · 91058 Erlangen · Germany

http://www7.cs.fau.de/

Implementation of Blockchain-Based and Grid-Friendly Local Energy Markets

Masterarbeit im Fach Maschinenbau

vorgelegt von

Matthias Seidl

geb. am 02. November 1993
in Kitzingen

angefertigt am

Lehrstuhl für Informatik 7
Rechnernetze und Kommunikationssysteme

Department Informatik
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuer:	**Prof. Dr-Ing. Reinhard German** **M.Sc. Jonas Schlund** **M.Sc. Moritz Meiners**
Abgabe der Arbeit:	**16. Oktober 2018**

Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.
Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Declaration

I declare that the work is entirely my own and was produced with no assistance from third parties.
I certify that the work has not been submitted in the same or any similar form for assessment to any other examining body and all references, direct and indirect, are indicated as such and have been cited accordingly.

(Matthias Seidl)
Erlangen, 16. Oktober 2018

Abstract

Germany is trying to face climate change by increasing its share of volatile renewable energy production and using energy as efficiently as possible. Currently, the energy grid is controlled by central authorities, which mainly manage conventional energy resources. However, this system is starting to face many challenges because of the increasing share of unpredictable energy production from renewables and decreasing share of conventional energy resources.

A more flexible system that acts self-sufficient in local areas and gives more power and responsibility to smaller- and medium-sized prosumers is a promising solution. This system can only be implemented if prosumers are convinced to act grid-friendly and acquire more flexibilities; it will help flatten the local supply/demand curves. By allowing them to profit from their helpful actions at the market, such a system can be achieved. Real-time supply and demand based energy trading, where prosumers can profit from flexibilities by acquiring energy when supply peaks and selling when demand peaks, leads to better prosumer engagement. Rewarding or penalizing grid affecting actions also leads to grid-friendlier behavior. For the information infrastructure, blockchain is currently discussed controversially. It offers numerous advantages, such as transparency, security, lower operation costs and its decentralized nature. However, it also comes with many technical renewals whose prospects and boundaries are still to be explored.

The goal of this thesis is to create blockchain-based, grid-friendly LEMs, where, in particular, the use of blockchain technologies as information infrastructure is reviewed, along with different approaches to achieve grid-friendly behavior. As a result, decentralized blockchain-based and grid-friendly Local Energy Markets (LEMs) are conceptualized and implemented as prototypes. Moreover, the implementation of markets and their agents is analyzed through different scenarios showing that it is possible to create blockchain based LEMs reacting in 30-second time-slots to current supply and demand. These markets are also able to contribute to the stability of the grid's voltage and frequency level, as well as, congestion management. The results show that LEMs with a higher local self-sufficiency and the tackling of growing grid operating expenses in more distributed approaches is possible.

Kurzfassung

Deutschland entgegnet dem Klimawandel mit der Erhöhung des Anteils an volatilen erneuerbaren Energien. Aktuell wird Strom vor allem durch konventionelle Kraftwerke erzeugt und das Stromnetz von zentralen Autoritäten kontrolliert. Jedoch kommt es für dieses System zu immer mehr Herausforderungen, da der Anteil an schwer vorhersehbaren, erneuerbaren Energiequellen zunimmt und der Anteil an konventionellen Energiequellen abnimmt. Ein flexibleres, lokales System, welches autark agiert und mehr Verantwortung an kleine und mittlere Prosumenten abgibt, stellt eine vielversprechende Lösung dar. Dieses System kann nur dann umgesetzt werden, wenn Prosumenten überzeugt werden, netzfreundlich zu handeln und zusätzliche Flexibilitäten zu aquirieren, was helfen wird, die Angebots- und Nachfragekurve abzuflachen. Dies kann erreicht werden, wenn es den Prosumenten ermöglicht wird, von ihren hilfreichen Aktionen zu profitieren. Echtzeit angebots- und nachfrageorientiertes Handeln von Strom, bei dem Prosumenten von mehr Flexibilitäten profitieren, indem sie Strom bei Überproduktion kaufen und bei Unterproduktion verkaufen, kann zu mehr Prosumenten Beteiligung führen. Durch die Belohnung und Bestrafung von netzbeeinträchtigenden Maßnahmen kann zudem netzfreundlicheres Verhalten erreicht werden. Blockchain wird aktuell viel und kontrovers diskutiert. Wegen den vielen Vorteilen, wie z.B. Transparenz, Sicherheit, niedrigere Kosten und der dezentralen Natur der Blockchain, stellt sie eine gute Option für die IT-Infrastruktur des neuen Systems dar. Jedoch kommt sie auch mit vielen technischen Neuerungen, wessen Möglichkeiten und Grenzen für diesen Anwendunsfall noch untersucht werden müssen. Das Ziel dieser Arbeit ist es blockchainbasierte, netzfreundliche, lokale Energiemärkte zu kreiieren und vor allem die Nutzung von Blockchain als IT-Infrastruktur zu untersuchen, genauso wie verschiedene Ansätze um Netzfreundlichkeit zu erreichen. Im Zuge dessen wird ein dazu passendes Konzept aufgestellt, die Märkte auf Blockchain-Basis prototypisch implementiert und die fertige Implementierung mit Hilfe von ausgewählten Szenarien analysiert. Als Ergebnis kann festgehalten werden, dass es möglich ist ein derartiges blockchainbasiertes System zu implementieren und, dass es auch zur Netzstabilität beitragen kann, jedoch ist vor allem die Skalierbarkeit kritisch zu sehen.

Contents

Chapter 1

Introduction

Humanity is facing its greatest environmental challenge ever seen. Over the last 150 years, we've changed the balance of our planet by mainly burning huge amounts of fossil fuels for energy production, resulting in a significant change of climate [1]. In order to tackle this rather negative development, Germany has committed to increase its share of renewable energy production by introducing the "Erneuerbare-Energien-Gesetz" (EEG) in 2000. Between 2000 and 2017, the share of renewable energy in Germany's electricity consumption has grown steadily from around 6% to 36% [2]. At the onset of 2018, Germany also lifted its renewable energy target from 50% to 65%, aiming to reach it by 2030 [3]. Besides increasing the share of renewables, it is also vital to increase energy efficiency and to cost-efficiently improve congestion management of transmission and distribution lines [4].

These ambitious goals, and the fact that the production of renewable energy is not controllable, make an entire restructuring of the current energy markets inevitable. The former top-down approach in which central institutions regulated the consumption and distribution of energy has to make room for a more flexible, local, and demand-controlling bottom-up approach. An approach where Local Energy Markets (LEMs) aim to be as self-sufficient as possible in order to avoid transmission and distribution bottlenecks and simplify the grid's operation organization. The self-sufficiency can only be achieved by adding more flexibilities (e.g. in form of batteries) to the market. An easy way to convince small- and medium-sized energy prosumers to invest in more batteries is to let them profit from the additional flexibilities through offering them supply- and demand-based real-time pricing. This allows the prosumers to acquire cheap energy at high supplies, store the energy in their batteries, and sell it at an expensive rate when demand peaks. Aside from the balance of demand and supply, many other factors affect the grid's stability. Instead of having complex structures where primarily central institutions control and manage the influencing factors of the grid's stability, every small- and medium-sized

grid participant should have the ability to contribute to and profit from the grid's stability it if they act grid-friendly. On the other hand, non grid-friendly actions should be penalized. This way a completely self-sufficient LEM can be realized.

In order to fully decentralize the energy markets, it is important that the underlying information technology also becomes decentralized. Blockchain technologies, which are currently experiencing a huge growth, are a promising infrastructure for the use case of LEMs. They offer many advantages, such as higher transparency, better security, an ability to fit the decentralized nature of energy production/-consumption, and no necessity for intermediaries. Through the use of blockchain technologies, the market will not have to rely or depend on any central authority. However, the prospects and boundaries of this technology for this specific use case must be explored.

The goal of this thesis is to create blockchain-based, grid-friendly LEMs, where, in particular, the use of blockchain technologies as information infrastructure is reviewed, along with different approaches to achieve grid-friendly behavior. To achieve this, a concept for different blockchain-based and grid-friendly LEMs, their participants (agents), and the underlying blockchain infrastructure is created. This concept is then implemented as a prototype with the help of blockchain technologies. Ultimately, different simulations are conducted in order to verify the prototype's functionality, compare the different implemented markets, and generally review the usage of blockchain technologies for such a use case.

Chapter 2

Fundamentals

The following sections will explain the fundamentals necessary to fully understand the concept and implementation, which are later shown in this thesis. The "Electricity Markets" section gives an overview of how energy markets, especially in Germany, currently work, and also explains how the grid's stability can be influenced. The "Blockchain Technology" section gives a complete overview of the fundamentals of blockchains.

2.1 Electricity Markets

Electricity markets differ from standard marketplaces due to electricity's unique characteristics. This section explains the main differences through a general overview, but also focuses on the influences that certain scenarios may have on the grid's stability.

2.1.1 General Overview

In general, every power grid can be split into three layers: an energy layer, an information layer and a market layer. All of these layers hold an important role within the grid. In addition to the layers, several parties interact in a highly complex manner in order to make the grid work.

The energy layer focuses on the actual power flow between the participants and has to take into account the special characteristics of electricity, such as the power flow, which is always continuous and goes through the path with minimal resistance. This means that the actual power flow doesn't always match the market information [5]. Additionally, it is currently not possible to store large amounts of energy cost efficiently or without large losses [6]. For this reason, all participants of a market should always produce as much as the market consumes. This, however, isn't

always possible. It is hard to control the generation of energy, especially since we are moving to more Renewable Energy Ressources (RESs). This is why the former top-down approach of managing and distributing energy has to make room for a bottom-up approach, where market participants actively influence the markets' supply and demand. [5]

The information layer, which controls the flow of information, is discrete and directed. A very important aspect of the information layer is its security of the information exchanged. [5]

Through the market layer, the participants of the grid are able to exchange money for energy. Before the liberalization of Germany's energy market, which started in 1998, the energy market was managed by the government. The centrally managed energy market led to many monopoles in the sector. As a result of the liberalization, more energy suppliers joined the market, causing it to become more competitive. Since then, consumers can freely choose their energy supplier and pay a standardized fee for using the grid. [7]

Currently, there are several different markets for several different time horizons and trading amounts. Aside from the trading of energy, ancillary services are also traded, such as primary control power [8]. The following Figure 2.1 illustrates the structure of Germany's electricity market:

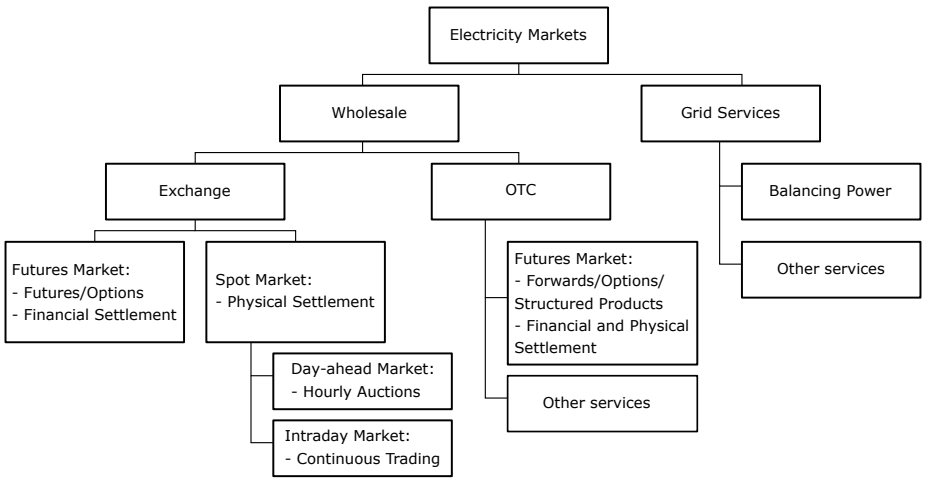


Figure 2.1 – Electricity Markets Structure [9]

On the Wholesale level, electricity is tradable through exchanges or Over the Counter (OTC). In the OTC market, direct bilateral trades are mostly executed between utilities and banks, and primarily made for standard trading products, such as futures, through individual contracts. The time horizon for these trades ranges from months to years. At the exchanges (e.g. EEX, APX, Nordpool), only standard

products (e.g. forwards with specific granularity) are traded through clearings. Participants need to be licensed with a specific exchange and have to pay entry fees (about 25,000 Euros), as well as yearly fees and fees per trade [10]. The time horizons vary based on the exchange type. For the future market, it ranges from weeks to months. For the day-ahead market, the time horizon is about two weeks, and for the intraday market the time horizon ranges from one hour to one day. [9]. Grid/ancillary services are mainly traded at the balancing power market. The ancillary services offered are frequency stability, voltage stability, black start capability, and system operation. [9]. Participants who want to offer their services have to meet certain qualifications which cannot be met by simple households [10].

The most relevant parties in Germany's liberalized electricity market, according to [8], are the following:

- **Transmission System Operator (TSO):** The TSO operates and maintains the long-distance transmission lines on a high voltage level. It has the responsibility to compensate the difference in supply and demand in the transmission system.
- **Distribution System Operator (DSO):** The DSO operates and maintains the medium and lower voltage electrical network, as well as their connection to the transmission grid.
- **Producer:** Producers are responsible for generating electric power.
- **Supplier:** Suppliers are responsible for selling energy to the customer. They can also be a wholesaler or independent trader.
- **Customer:** Customers can freely choose the supplier it wants to purchase electricity from.

At the moment, average households and small-scale renewable power producers have fixed prices for the consumption and sale of of energy. As a result, they cannot participate in the grid's stability by offering flexibilities, or benefit as consumers or power producers through production and consumption peaks. Even though households contain more and more smart devices, smart home management systems, and batteries, it is still not possible for them to benefit from this technological process by smart interaction with the grid. In order to let the whole grid benefit from this technological progress, markets have to adapt so that they can offer more flexible prices to households, depending on the current supply and demand, and possibly even on the grid's state.

2.1.2 Influences on Grid Stability

Many factors influence a grid's stability. In this thesis, we focus on the voltage stability, frequency stability, and congestions of transmission lines. These factors will

be explained briefly in the following.

Frequency

The nominal frequency (within the European power system) is 50 Hz and remains the same value at every node in the same power system [8]. The frequency is influenced by an imbalance of supply and demand. In the case of high demand, the frequency lowers, and in the case of high supply, the frequency rises. In order to prevent blackouts and other negative influences on the grid, the frequency has to stay within a small range of tolerance (±0.2 Hz). In the current energy market, it is the TSOs responsibility to ensure frequency stability through balancing services, reserve capacity, frequency control, and reactive power supply. The most important tools for frequency stabilization are the primary control energy (also known as Frequency Containment Reserve (FCR)), which is defined by Article 3(2)(6) of the EU Network Code on System Operation as "the active power reserves available to contain system frequency after the occurrence of an imbalance" [11], and the secondary control energy (also known as Frequency Restoration Reserve (FRR)), which is defined as "the active power reserves available to restore system frequency to the nominal frequency" [12]. The tertiary control is also important, but not used as often. They all differ in regard to their activation speed. In the case of a deviation greater than 10 mHz from 50 Hz, FCR becomes activated automatically within seconds and will be activated completely at a deviation of 200 mHz or more. After 120 seconds, the FCR is replaced by FRR, and in the case of an extended disruption, FRR control is supported by tertiary control after roughly 15 minutes. In order to be able to offer these control mechanisms, the TSO has to procure flexibilities from participants in the grid on a weekly (primary and secondary control) or daily basis (tertiary control) [8]. The following Figure 2.2 illustrates the changes of control energy:

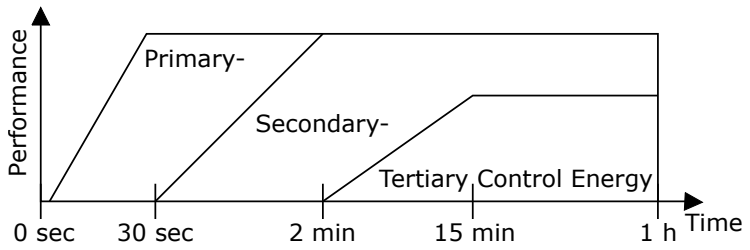


Figure 2.2 – Chronological Process of the Activation of Control Energy [8]

Because of the rise of RES, supply is becoming more and more distributed, uncontrollable, and harder to predict. For this reason, the TSO has to get ahold of more and more control energy, especially from traditional power plants and large industrial loads. This shows the limits of integrating RESs into the current energy

system. [13]

Voltage

The voltage of a power system can vary based on the load in the system. A high load leads to a low voltage, while a high power feed-in leads to the inverse, a high voltage. These voltage changes, if severe enough, can cause damage to electrical equipment that requires steady voltage. To prevent this from happening, voltage regulators, or load tap changers, are incorporated into a power transformer. These devices are capable of increasing the system voltage when it is too low and reducing the system voltage when it is too high, in order to maintain a steady voltage. [14]

Voltage differs from node to node and usually only influences the lower voltage network. The higher generation through RESs, especially Photovoltaic (PV), and the two-way power flow have introduced new uncertainties and technical challenges in voltage control. High penetration of renewable generation can potentially result in voltage peaks if the load is too small. [15]

Congestion

Congestions occur when transmission networks fail to transfer power based on the load demand. This can lead to serious effects on power systems [16]. In Germany, a number of intense energy-consuming industries are located in the south, yet a lot of energy is generated in the north, particularly through large, offshore wind plants. Moreover, there's been a phase-out in nuclear power plants in the south, which has led to a lower supply in the area [17]. This geographical imbalance between demand and supply can lead to congestion in the transmission network.

In order to avoid congestions, TSOs take re-dispatch measures (e.g. reducing power generation of conventional power plants in the north, shutting down wind turbines temporarily, or producing more energy in the south in less efficient, conventional power plants). These re-dispatch measures are expensive, and with the increasing amount of RESs it is also becoming more expensive for the German energy consumers. This is because these re-dispatch measures are passed onto consumers as part of the grid-fee that households pay via their electricity bill. From 2014 to 2015 alone, the amount paid for re-dispatch measures rose from 183 million euros to 420 million euros and is expected to rise to 1 billion euros by 2020. [18]

In a nutshell, the current energy market is facing many challenges, mainly because of the increasing amount of RESs. In order to overcome these challenges, new methods have to be implemented and tested, thoroughly.

2.2 Blockchain Technology

This chapter explains the motivation behind the development of blockchain technologies, the functionality of the Ethereum blockchain versus Hyper Ledger, and the architecture of the Ethereum blockchain, as well as highlight the software and tools that are used in this thesis.

2.2.1 Development History

The article, "Bitcoin: A Peer-to-Peer Electronic Cash System", was published in November 2008 under the pseudonym Satoshi Nakamoto [19]. It is Bitcoin's so-called Whitepaper which describes not only the currency, but the architecture and functionality of the Bitcoin network as well. Bitcoin was the first ever use-case of blockchain and has been online since January 2009 [20]. It is essentially a cryptocurrency which isn't administered centrally, like every other currency at this point in time. The architecture of the Bitcoin blockchain is designed to be decentralized. Anyone can participate in the network through validating blocks; a process called mining. Furthermore, the entire source code is open-source and every transaction on the Bitcoin network is visible to the public eye. This is why blockchain technologies are considered a subgroup of Distributed Ledger Technologiess (DLTs). The Peer-to-Peer architecture makes it possible for every payment to go directly from one account to another without any central financial institution. Through the use of digital signatures, hashing, and proof-of-work as a consensus algorithm, Bitcoin prevents double-spending (spending the same digital token more than once) and delivers a practical solution for the Byzantine Generals problem (an agreement problem, where it is essential to make sure that all participants come to the same agreement, without communicating through the same messenger [21]).

In 2014, Vitalik Buterin published Ethereum's Whitepaper, "Ethereum: A Next Generation Smart Contract and Decentralized Application Platform", in the Bitcoin Magazine. Ethereum is another cryptocurrency similar to Bitcoin, but its underlying technology and blockchain is far more advanced. Bitcoin only allows transactions of tokens, whereas Ethereum provides a built-in Turing-complete virtual machine, the Ethereum Virtual Machine (EVM), which can execute bytecode. Users can deploy Smart Contracts on the Ethereum blockchain, allowing every node in the network to interact with the deployed Smart Contract. The miners in the Ethereum network validate the blocks of the blockchain while also running the Smart Contracts. This way, any kind of logic can be implemented and executed in a completely decentralized manner. These programs are called Decentralized Apps (dApps). [22]

Aside from Bitcoin and Ethereum, there exist many other blockchain technologies. The most important one pertaining to this thesis is Hyperledger Fabric, which is part

of the Hyperledger project that started in December 2015 by the Linux Foundation and was originally contributed by IBM. It, too, allows Smart Contracts, but what sets it apart is the fact that it is a permissioned blockchain (this term will be explained in the Terminology section) that assigns different roles to different participants, which is especially valuable for business applications. [23]

As of July 2018, Bitcoin has become the cryptocurrency with the highest market cap, reaching 125 billion USD. Ethereum has the second highest market cap with 51 billion USD. [24]

2.2.2 Terminology

In order to better understand the rest of this thesis, this section shows the fundamental terminology for blockchain technologies.

In general, blockchains are hash pointer based data structures composed of blocks [25]. The blockchain itself can also be seen as a Distributed Ledger (DL), which stores the same data at every node that is participating at the blockchain (redundancy). Because of their DL and decentralization, the consensus mechanism plays a central role for blockchains. Bitcoin and Ethereum are currently using Proof of Work (PoW) as a consensus mechanism. Other consensus mechanisms also exist, but are more feasible for different types of blockchains. In order to understand this, the terms permissioned and permissionless must be explained first.

"In permissionless blockchains anyone can participate in the network and consensus mechanisms such as proof-of-work (PoW) are needed to arrange that the state of the network is not corrupted. PoW means that computational work has to be done in order to mine a new block. Thus an attacker would need more than 50 % of the total computational power of the network in order to corrupt it. In permissioned blockchains more energy efficient concepts like proof-of-authority (PoA), where only validators are allowed to mint (produce without work) new blocks. With PoA the network cannot be manipulated as long as 50 % of the validators are honest. Furthermore blockchains can be private or public depending on if anyone can read the data from the blockchain." [26]

The aforementioned PoW is one way of finding a consensus within a permissionless blockchain. Miners are hashing the input of the entire block repeatedly, always adding a different nonce until they find a hash value that matches the search criteria. The miner that finds the fitting nonce first receives a reward. The downside of this mechanism is that many people compete for the mining reward which leads to an excessive use of computing power and electricity. Another way to reach a consensus is Proof of Authority (PoA), which doesn't rely on brute computing power, but on authority nodes that have the power to validate blocks [27]. PoA is only feasible for permissioned blockchains. Authority nodes leverage their identities,

making it essential that the validation of these authority nodes fulfill the following requirements [28]:

- Their identities need to be formally identified on-chain with the ability to cross-reference these identities through reliable data available in the public domain (such as a public notary database).
- Eligibility to become a validator must be difficult to obtain in order to ensure that the long-term prospective position of the validator is one of clear incentive, both financially and reputationally, to remain an honest validator.
- There must be complete uniformity in the process of establishing validators.

PoA chains cannot be seen as decentralized like PoW chains because of the previously mentioned authority nodes. Additionally, the security of the whole PoA chain relies on trust in the validators, who are subjected to an outside influence from third parties that may have interest in seeing the network fail. Nevertheless, they are much more efficient in terms of transaction times and overall network consensus [28].

Another possible consensus mechanism is Proof of Stake (PoS), which can potentially be used in permissioned and permissionless blockchains, but is not part of this thesis, and therefore not explained in more detail.

In summary, blockchains need a feasible consensus mechanism because of their decentralized nature. PoW works well for permissionless chains, but is limited in its performance, while PoA works well for permissioned chains and has an enhanced performance in comparison to PoW.

The following Table 2.1 briefly explains the terms relevant to blockchain technologies that are necessary in understanding the following section, containing a more detailed explanation of blockchain architecture:

Term	**Explanation**
Address	Addresses are used to send or receive transactions on the network. An address usually presents itself as a string of alphanumeric characters. Smart Contracts have their own addresses, as well as user accounts.
Block	Blocks are packages of data that carry permanently recorded data on the blockchain network. Every block, besides the genesis block, is linked to the block preceding it through its hash value.
Block Height	The block height is the number of blocks connected on the blockchain.

Block Reward	The block reward is a form of incentive for the miner who successfully calculates the hash in a block during mining. The verification of transactions on the blockchain generates new coins in the process, and the miner is rewarded a portion of those.
Block Size	The block size is the amount of data that can be stored in one block.
Block Time	The block time is the time it takes on average until a block is minted or mined.
Consensus	Consensus is achieved when all participants of the network agree on the validity of the transactions, ensuring that the ledgers are exact copies of each other. Consensus can be achieved by many different mechanisms, such as PoW, PoA, etc.
DAO	Decentralised Autonomous Organizationss (DAOs) can be thought of as corporations that run without any human intervention, that surrender all forms of control to an incorruptible set of business rules.
dApp	A dApp is an application that is open source, operates autonomously, has its data stored on a blockchain, incentivises in the form of cryptographic tokens, and operates on a protocol that shows proof of value.
Distributed Ledger	DLs are ledgers in which data is stored across a network of decentralized nodes. A distributed ledger does not have to have its own currency and may be permissioned and private.
Ethereum Virtual Machine	The EVM is a Turing complete virtual machine that allows anyone to execute arbitrary EVM Byte Code. Every full Ethereum node runs the EVM to maintain consensus across the blockchain.
Fork	Forks create an alternate version of the blockchain, leaving two blockchains to run simultaneously on different parts of the network.
Gas	Every transaction on the Ethereum blockchain consumes a certain amount of gas. The amount of gas depends on the complexity of the transaction. Gas is paid to the miner of the block.

Genesis Block	The Genesis block is the very first or the first few blocks of a blockchain.
Hash	A hash is the result of performing a hash function on the output data. This is used for confirming coin transactions.
Mining	Mining is the act of validating blockchain transactions. The necessity of validation warrants an incentive for the miners, usually in the form of coins.
Node	A node is a copy of the ledger operated by a participant of the blockchain network. Nodes can either copy the whole ledger (full-node) or only copy information that is necessary for their interaction with the blockchain (light-node, fast-node).
Peer-to-Peer	Peer-to-Peer (P2P) refers to the decentralized interactions between two or more parties in a highly-interconnected network. Participants of a P2P network deal directly with each other through a single mediation point.
Private Key / Public Key	A private key is a string of data that allows you to access the tokens in a specific wallet. They act as passwords that are kept hidden from anyone but the owner of the address. The public address is derived by hashing the public key. They act as addresses that can be published anywhere, unlike private keys.
Proof of Authority	PoA is a consensus distribution algorithm that requires authority nodes, which are able to validate transactions.
Proof of Stake	PoS is a consensus distribution algorithm that rewards earnings based on the number of coins you own or hold. The more you invest in the coin, the more you gain by mining with this protocol.
Proof of Work	PoW is a consensus distribution algorithm that requires an active role in mining data blocks, often consuming resources, such as electricity. The more work you do or the more computational power you provide, the higher is the chance that you are rewarded with more coins.
Smart Contract	Smart contracts encode business rules in a programmable language onto the blockchain and are enforced by the participants of the network.
Solidity	Solidity is Ethereum's most popular programming language for developing smart contracts.

Testnet	The testnet is a test blockchain used by developers to prevent expending assets on the main chain.
Transaction Fee	All cryptocurrency transactions involve a small transaction fee. These transaction fees add up to account for the block reward that a miner receives when he successfully processes a block.

Table 2.1 – Blockchain Terminology [29, 30]

2.2.3 Architecture

In this section, the general architecture of a blockchain, as well as Ethereum's blockchain, which is used later for the market implementation, is explained briefly.

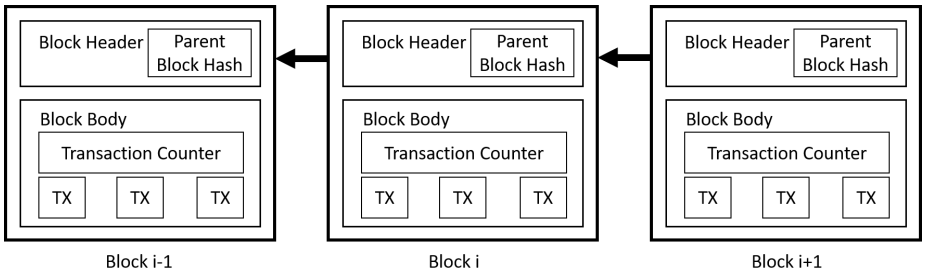


Figure 2.3 – An example of a blockchain which consists of a continuous sequence of blocks [27]

Figure 2.3 shows three blocks that are all linked to their preceding blocks through the block hash information. In addition to the parent block hash, the block header contains the block version, the Merkle Tree Root hash, a timestamp, nBits, and the mining nonce. Aside from the block header, a block also stores multiple transactions and counts them within the block body. Figure 2.4 shows the complete structure of a block.

As previously mentioned, every block points to its preceding block through the 256-bit parent block hash, but the first block of a blockchain, the genesis block, has no parent block. The block version contains the rules of block validation. The Merkle Tree Root hash stores the hash value of all transactions in the block. The timestamp stores the current time in seconds since January 1, 1970. The nBits information holds the target threshold of a valid block hash and the nonce is the value which a miner iteratively changes in order to find a valid block hash [27].

The block body contains all the transactions for this block. The number of transactions depends on the maximum block size and the size of each transaction. Bitcoin's block size is 1 Megabyte. Ethereum has a gas limit rather than a block

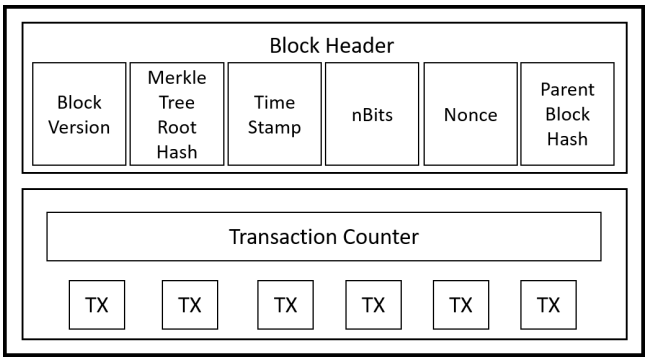


Figure 2.4 – Block structure [27]

size. Every transaction in the Ethereum network uses a specific amount of gas. This amount depends on the complexity of the transaction's computation - the more complex, the more gas is necessary. The gas expenses for every operation are listed in Ethereum's Yellow Paper [31]. Adding two numbers, for example, costs 3 gas, while the cost of calculating a hash is 30 gas and sending a transaction costs 21,000 gas. Bitcoin's block size is constant in contrast to Ethereum's gas limit, which can be changed by the miners. Since gas is used on the Ethereum blockchain, blocks in the Ethereum blockchain also have both the gas-limit and gas-used (total gas used in all transactions in the current block) information in their block header [22].

For the Bitcoin blockchain, every block is expected to take about 10 minutes to be mined. Because Bitcoin is using PoW as a consensus mechanism, this is only an estimated time; the validation of the block depends on the luck of the miners. But if more miners try to validate a block, the blockchain regulates its difficulty to maintain the average block time around 10 minutes [27]. The public Ethereum blockchain, which also works with PoW, functions in a similar manner. One difference is that the average block time is much lower, currently averaging at 15.96 seconds [32] (on 14.09.2019). But Vitalik Buterin plans to have an expected block time of 12 seconds in the future [33].

The performance of blockchains is often measured in Transactions per Second (TPS). At the moment, Bitcoin has about 3-7 TPS and Ethereum has about 15-20 TPS [34], but both of these values can increase significantly in the future through the introduction of concepts such as state channels and parallel chains. Additionally, the quantification TPS (which is often used by critics who compare blockchains to centralized services, such as Paypal, with 450 TPS) isn't correct in Ethereum's case, because every transaction varies in its complexity and use of gas. For this reason, the Energy Web Foundation (EWF) prefers to quantify the throughput on blockchains with smart contracts in Stuff per Second (SPS), rather than TPS [35].

2.2.4 Comparison of Different Blockchain Technologies

For the purpose of this project, we need a blockchain technology that allows the deployment of smart contracts. The ideal blockchain should also offer other characteristics that make it feasible for this project:

- **Governance:** The blockchain technology should be decentralized and as independent as possible.
- **Consensus:** The consensus mechanism should be reliable and as time-discrete as possible (PoW not feasible).
- **Network:** The access to information within the network should be restricted.
- **Smart Contracts:** The programming language used should be easily readable for participants.
- **Ecosystem:** The larger and more mature the ecosystem, the better.
- **Scalability:** The ability to handle a growing number of participants without any negative consequences.

In this section, a private Ethereum instance will be compared to Hyperledger Fabric based on the various characteristics mentioned above.

Regarding governance, Ethereum is governed by both the Ethereum developers and the Enterprise Ethereum Alliance. Hyperledger Fabric is governed by the Linux Foundation and IBM. Both projects are completely open source. Ethereum's goal is to offer a decentralized virtual machine for applications that are distributed in nature, whereas Hyperledger Fabric aims to be as flexible as possible and has a very modular architecture that makes it feasible for business. [36]

The consensus mechanism is the most important part of every blockchain. Hyperledger Fabric offers configurable consensus through its orderer service [37]. This means that it can, compared to other blockchains, easily switch between different consensus algorithms. This is made possible by the Practical Byzantine Fault Tolerance (PBFT), which makes sure that the transactions are in the correct order. Additionally, Hyperledger Fabric uses endorsement policies and an array of validation steps. In a private Ethereum instance we can also choose between different consensus mechanisms, but it's not as easily exchangeable. During the making of this project, Ethereum offered PoW and PoA as consensus mechanisms, but in its roadmap, it is outlined that with the "Constantinople" release of Ethereum, it will slowly introduce its own PoS consensus mechanism, called Casper [38].

Ethereum was designed to be a publicly accessible blockchain. However, as touched on above, it is possible to create private Ethereum instances [39]. These are networks that are created by a network starter, in which participants are required

to have an invitation in order to join the network. Hyperledger Fabric was already designed for this purpose. It allows private permissioned blockchains and it even offers roles that make it possible to restrict the visibility of transaction [36]. This means that only participants obtaining the permission necessary can view these transactions. This is a very strong feature that isn't available on Ethereum. In the Ethereum network everyone can see all transactions.

Both Ethereum and Hyperledger Fabric allow the creation and deployment of smart contracts. Even though Hyperledger Fabric has slightly better fitting characteristics for this project, we decided to use Ethereum for the implementation of this prototype, due to its bigger development environment and community.

Chapter 3

Related Work

There has been a fair amount of research on market designs and agent strategies for LEMs, but it is only since the beginning of 2017 that researchers have started reviewing blockchains as the central building block of decentralized LEMs. This chapter highlights the most important research in the areas of market design, agent strategy, and blockchain for LEMs.

3.1 Market Designs for Local Energy Markets

The goal of a good market design is to find an equilibrium price at which the biggest amount of energy can be traded locally. The literature review showed three main strategies: random pairwise trading, price functions, and auctions.

3.1.1 Random Pairwise Trading

In their paper, BLOUIN et al. propose a P2P market design with random and anonymous pairwise meetings with individual bargaining [40]. The goal of this design it to get rid of any auctioneers and avoid the central aggregation of information. Sellers and buyers become randomly matched and bargain prices via an anonymous channel. If they cannot come to an agreement, seller and buyer remain in the market and become matched again with another counterpart. If they do come to an agreement, they transact and either stay in the market (if they still have electricity to trade) or leave the market. This mechanism leads to an individual price for every seller-buyer agreement. The amount of energy traded locally reaches, at least, the amount of energy traded locally through order book markets/auctions, as the P2P market finds all trades the auction market facilitates plus a number of trades facilitated by favorable P2P matching [41].

3.1.2 Price Functions

MIHAYLOV et al. propose a market that relies on price functions. These price functions are controlled by the DSO and are determined every 15 minutes based on the total production and consumption, which will be delivered from smart meters, the new prices for buyers, and sellers. Instead of using the public grid prices as boundaries, the range of these price functions is larger. The price for consuming energy can, for example, equal 0 during times when energy production is much higher than consumption, which can be seen in the following formulas. [42]

The price function $g(x, t_p, t_c)$ for producers is defined as:

$$g(x, t_p, t_c) = \frac{x \cdot q_{t_p=t_c}}{e^{\frac{(t_p - t_c)^2}{a}}} \tag{3.1}$$

And the price function $h(y, t_p, t_c)$ according to which consumers pay, is defined as:

$$h(y, t_p, t_c) = \frac{y \cdot r_{t_c >> t_p} \cdot t_c}{(t_c + t_p)} \tag{3.2}$$

where:

x	=	amount of energy produced by participant
y	=	amount of energy consumed by participant
t_p	=	amount of total supply
t_c	=	amount of total demand
$q_{t_p=t_c}$	=	maximum rate at which producers are rewarded for their input energy
a	=	scaling factor (in case $t_p \neq t_c$)
$r_{t_c >> t_p}$	=	maximum cost of energy delivered when supply is low

This approach is not completely decentralized, since the DSO and energy supplier still play a major role in the price formation. However, the price functions achieve to incentivize agents, balancing supply and demand out of their own self-interest. This will flatten the consumption and production peaks. Additionally, this approach is scalable; newly joining agents do not increase the complexity of the energy trade. [42]

VYTELINGUM et al. uses a Continuous Double Auction (CDA) but includes price functions in order to handle the congestion of transmission lines. Owners of transmission lines can charge the market participants based on the amount that they are transporting through their transmission lines. Simultaneously, participants can receive money for generating a counterflow in the transmission lines. [43]

3.1.3 Auction Mechanisms

More common than using random pairwise trading or price functions is the use of auction mechanisms. In the case of LEMs, the Double Auction (DA) proves the most popular market design. At the start of each trading period, an auctioneer collects the bids from potential buyers and the asks from potential sellers. Then, the auctioneer chooses, based on a predefined mechanism, the clearing price for the market. Buyers that bid more than the Market Clearing Price (MCP) buy at the MCP, and sellers that asked for less than the MCP sell at the MCP. Buyers try to get a low price while competing against other buyers, but sellers strive for the opposite. In their case, the aim is trying to get higher prices while competing against other sellers. [44]

The following Figure 3.1 illustrates the supply and demand curves for a double auction:

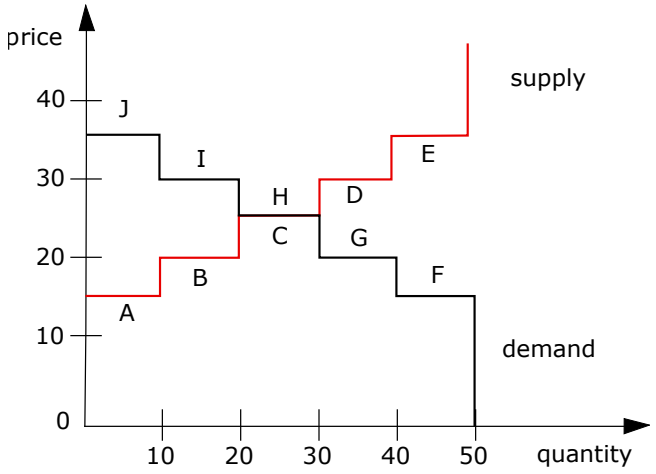


Figure 3.1 – Illustrative supply and demand curves for a double auction. [44]

The price at which supply equals demand is known as the equilibrium price, and in the case of the market in Figure 3.1, it is a price of 25.

'In a periodic or Discrete Time Double Auction (DTDA), clearing happens during some fixed period of time after the auction starts. The Continuous or open-outcry DA, in contrast, does not have a specified time for clearing. In institutional trade-determination, the institution effectively has to try to clear the auction (which means find a bid and ask that cross) whenever a new bid or ask is made by a trader.' [44]

Based on the Myerson-Satterthwaite theorem, an ideal auction mechanism satisfies the following principles: [45]

- **Individual Rationality (IR):** The participants have an initial incentive to participate.

- **Weak Balanced Budget (WBB):** The auctioneer does not have to subsidize the trade with his own money.
- **Nash Equilibrium Incentive Compatibility (NEIC):** The participants are incentivized to report their true value if the other agents also report their true value
- **Pareto Efficiency (PE):** The item is given to the participant who values it the most

Unfortunately, according to the Myerson-Satterthwaite theorem it is not possible to achieve all these requirements in the same auction mechanism. Achieving 3 out of these 4 mechanisms is the maximum. The mechanisms important for this thesis are the Average and the Vickrey Clarke Groves (VCG) mechanism. [45]

The **Average** mechanism follows the following steps [44]:

- Order the buyers in decreasing order of their bid: $b_1 \geq b_2 \geq ... \geq b_n$
- Order the sellers in increasing order of their asks: $s_1 \leq s_2 \leq ... \leq s_n$
- Find the the index k where $b_k \geq s_k$
- Set the price at the average of the values at index k: $p = (b_k + s_k)/2$
- The first k sellers can sell their goods to the first k buyers.

This mechanism satisfies the principles of IR, WBB, and PE, but not NEIC. This is because buyers have an incentive to report lower values, while sellers have an incentive to report values higher than what they are actually willing to pay/sell for. [44]

The **VCG** mechanism, on the other hand, satisfies the principle of NEIC through subsidizing trades [44]:

- Steps 1-3 from Average mechanism (Natural Ordering)
- The first k sellers give the goods to the first k buyers
- Each buyer pays the lowest equilibrium price $max(s_k, b_{k+1})$
- Each seller receives the highest equilibrium price $min(b_k, s_{k+1})$

This mechanism is not WBB, because the auctioneer has to pay for the difference between the lowest and highest equilibrium prices. [44]

For LEMs both, DTDAs and CDAs have been tested with different mechanisms. As previously mentioned, VYTELINGUM et al. use an average CDA for day-ahead

trading together with price functions for congestion and an online balancing mechanism, in order to cope with unforeseen supply and demand in real-time. They chose the CDA because previous studies have found them to be highly efficient in similar systems. They also made use of the New York Stock Exchange (NYSE) quote-accepting policy: only bids and asks that improve themselves are accepted in the market (buyers cannot decrease bid price and sellers cannot raise ask price). Limit orders (buy and sell with price constraint) and market orders (buy or sell exact quantity at any price) are possible. Market orders are usually cleared immediately if there are enough unmatched offers in the order book. If there aren't enough orders, they are placed on top of the order book. Information about the market state is made public to all participants through the bid and ask order books. [43]

In [46], ILIC et al. review the NOBEL project, which implemented and assessed a local energy market in a smart neighbourhood in Algine, Spain. Discrete, fixed-sized time slots of energy are traded at an average CDA market, where the order book and the last transaction prices are made public. In order to participate in the market, consumers and producers have to predict the amount of energy they want to buy or sell for the next time slot. These orders and the predictions accompanying them can be adjusted, as long as the order didn't get cleared. For simplicity's sake, prediction errors were not considered in the evaluation. Orders that received partial matches stayed in the order book with the remaining (unmatched) quantity.

TAN et al. conclude that "... the CDA potentially provides both economic efficiency and scheduling efficiency; it is thus a promising candidate for Grid resource allocation". [47] The DTDA, unlike the CDA, clears itself in distinct time intervals, which raises the question of what the best suitable time frame for the transaction period is, considering that the duration has certain effects on the feasibility of this auction type and influences the possibilities that if offers for the grid. [47]

In [48], MENGELKAMP et al. evaluate the Brooklyn Microgrid (BMG) as a case study. The BMG, run by the startup LO3 Energy, consists of a microgrid energy market in Brooklyn, New York with participants in three different distribution grid networks. It is a local, blockchain-based energy market that enables its participants to trade energy P2P with their neighbors, aiming to reduce grid issues resulted by severe weather events or RES. They are aware that local markets are only beneficial to their participants as long as the average energy price is lower than the external grid. The implemented market mechanism is an average DTDA with 15-minute time slots and the option to choose the preferred energy source (e.g. local renewable energy). Consumers that are not successful in the market can be supplied by additional energy sources (e.g. hyper-local energy or traditional, "brown" energy) or, eventu-

ally, by the public grid. Additionally, it is planned that other market mechanisms (e.g. pay-as-you-bid order book in which each transaction may have an individual transaction price) will be tested and evaluated.

Another interesting project is the Landau Microgrid Project (LAMP), in which 20 households (15 consumers and 5 prosumers) are connect via blockchain and enabled to trade their electricity locally. It is another use case for an average DTDA with 15-minute time slots. This market does not have a goal to balance energy or support the grid's stability in any way. These tasks are ensured by the public grid. It is solely an energy trading platform for the participants. LAMP is run by a cooperation of LO3 Energy and Energy Suedwest AG. [49]

The aforementioned market designs deliver a good foundation for this thesis. They either focus on fair and efficient trading, decentralization, or grid stability. In this thesis we incorporate all of these characteristics in one market design.

3.2 Agent Strategies for Local Energy Markets

In order to cope with the new challenges through RESs, decentralized, autonomous systems that are self-organizing and achieve high levels of efficiency are necessary [43]. Market-based approaches have been particularly successful at achieving high levels of efficiency when having to deal with large decentralized systems composed of self-interested agents, also known as Multi-Agent System (MAS) [50].

Agents are software programs that act as representatives of market participants (e.g. households). They interact with each other in order to accomplish an individual (e.g. buy cheap energy) or collective task (e.g. secure the grids stability). Because these agents are self-interested, they cause electricity consumption to shift towards moments of low electricity prices, and production towards moments of high prices, which ultimately flattens the demand/supply curves, having a positive impact on the grids stability. [5]

Agents are generally divided into Zero Intelligence (ZI) agents, who bid random numbers within a pre-arranged price range, and intelligent bidding agents, who can bid on either pre-defined rules or complex algorithms and data. The data could either be gathered from the market or be information about the current grid state (voltage, frequency, traded volume). [43]

LAMPARTER et al. see the MAS as a fitting solution for the existing challenges in LEMs, because of the decentralized nature of these challenges. They propose intelligent agents that implement a generic bidding strategy, governed by local policies which represent user preferences or constraints of the devices controlled by

the agent. These policies should ensure a high degree of autonomy, while making sure that the agents behave within a predefined action space. Policies can be added and removed at runtime, allowing the strategy to be adapted dynamically. The actions of the agents are based on a three layer structure. Agents gather information within the information layer (market state, agents' state, environment state). This pre-defined information is then combined with the user or appliance policies within the knowledge layer, which then returns the actions that are admissible or forbidden in order to constrain the strategy space of an agent in the bidding process. Ultimately, the agents choose the best action to take in the behavioral layer based on a ranking of the strategies according to the agents' preferences. [51]

In their paper, Trading on Local Energy Markets: A Comparison of Market Designs and Bidding Strategies, MENGELKAMP et al. compare P2P markets with closed order book DTDA markets. ZI agents and intelligent bidding agents are also reviewed for each market design. According to the results, the intelligent P2P market performs better than the other scenarios. It has the highest self-consumption, thus causing the smallest amount of money to leave the LEM. Furthermore, in all four scenarios, the prices within the LEM were, on average, lower than the public grid tariff, making participation in the LEM profitable for market participants. [41]

The implementation of intelligent bidding agents is not the focus of this thesis, however, the literature review showed that agents are necessary for a meaningful implementation and simulation of a LEM. For this reason, and for comparison's sake, both, ZI and intelligent bidding agents will be implemented.

3.3 Use of Blockchains for Local Energy Markets

The last two sections review related work in terms of market design and agents for LEMs. This section is about the use of blockchains in LEMs. It lists relevant, current implementations of blockchain based LEMs, as well as other projects that are supporting the use of blockchain in the energy sector, such as the EWF.

3.3.1 Existing Blockchain-Based Local Energy Markets

Blockchain-based LEMs are being reviewed by many researchers and startups at the moment because of their potential to offer many benefits over centralized LEMs:

- Blockchain technology makes microgrids more resilient by establishing trust between the involved agents, especially with respect to financial payments and electricity delivery. [48]
- Through the use of blockchain, there is no need for central intermediaries. [48]

- The generation and consumption of renewable energy is directly transferable into virtual coins. [48]
- The attainable degree of security is higher than in traditional, centralized trading platforms. [48]
- Blockchains offer a distributed software architecture, as well as a decentralized consensus mechanism, that match with the decentralization of energy generation. [49]

As mentioned earlier, the BMG and LAMP are blockchain-based implementations of LEMs. The BMG was the first real-life implementation of a blockchain-based LEM. Their energy market platform is based on a private blockchain using the Tendermint protocol. No detailed information is given about the blockchain technology that they used. After a three-month test trial, MENGELKAMP et al. conclude that private blockchains are suitable information systems for LEM with only small drawbacks: "An information technology that uses vast amounts of energy contradicts the sustainability principles of microgrid energy markets." No technological evaluation of blockchains as information and communication systems, with focus on scalability and robustness, was done in this project so far. [48]

Similar to the BMG implementation, the LAMP also makes use of a private blockchain, but once again, without further explanation of which blockchain technology was used [49].

HAHN et al. implemented a simple smart contract for a transactive energy auction on an Ethereum based private blockchain with Go-based Etherum nodes (geth). The work only demonstrates a proof-of-concept with a small number of agents and a short testing period. Additional research is necessary in order to explore the scalability and performance of the used blockchain technology. [52]

Powerledger, an Australian company that focuses on the use of blockchain technology in the energy sector, is working on multiple blockchain-based energy trading concepts. They introduced a hybrid public and consortium blockchain approach. Their tokens, POWR tokens, which are standard ERC20 tokens, can be traded on the public Ethereum network. Power Ledger also currently uses the EcoChain blockchain, a private PoS, low-power blockchain developed in-house and live-tested. This chain offers an additional token, the Sparkz token. However, in order to be able to handle the high transaction volume of P2P energy trading, Power Ledger has already commenced the transition to a modified, fee-less Consortium Ethereum network, while retaining its existing Ecochain system benefits. They also plan to increasingly utilize state channels to handle the high frequency nature of energy transaction settlements.

Some of their operational and conceptual applications include P2P trading, grid management, wholesale market settlement, autonomous asset management, electric vehicles, carbon trading, transmission exchange, etc. [53]

Oli Systems, a German startup offering technologies that support the German "Energiewende" (a change towards more sustainable energy sources), is also working on a blockchain-based energy trading platform. The first prototype deploys several smart contracts on a private Ethereum instance via go-ethereum (geth), allowing trading at a DTDA with 15-minute time slots. Additionally, a simple grid fee is implemented for distribution (trading locally) and transmission (trading outside of the own LEM). It is calculated for every auction period and charges based on the relative load. [54]

In order to have access to all implemented functions, a total of six smart contracts need to be deployed. These are briefly explained in the following Table 3.1:

Name	**Main Features of the Smart Contract**
Oli Coin Figure A.1	The OliCoin smart contract's purpose is to introduce a new token to the market, which is used as currency for energy trading. Through the functions of this smart contract, this token can be transferred between accounts and valid contract addresses can even set balances for addresses. Contract addresses (such as DaughterAuction or ParentAuction) need to be registered here by the contract owner in order to be allowed to set balances. Balances of addresses can also be returned. Even though this token is used for standard transactions, no token standards, such as ERC20, have been used.
Oli Origin Figure A.2	The OliOrigin smart contract functions as a gateway to the market. New market participants can be registered here by the contract owner. The information necessary for a new participant includes the account address (public key), longitude, latitude, trafo number, circuit number, the energy type (e.g. photovoltaic, wind, consumer etc.), and the trafo peak-load (which is only necessary if the energy type equals 8, signaling that a trafo is added). Lastly, the contract also offers a number of getters.

Daughter Auction Figure A.3	The DaughterAuction contract is a core contract within this system. It implements a double auction. Participants can send their bids and asks together with amount and price information, but only for their own trafo. Additionally, the contract includes a breakEven function thay finds the MCP and changes the balances of all successful consumers and producers according to their offered amount and bidding price, after adding a grid fee. A reset function clears all variables and, afterwards, new bids can be sent for a new auction period. The breakEven and reset function have to be called from outside.
Parent Auction Figure A.4	This contract is very similar to the Daughter Auction. The idea behind it is that participants who were not successful in the Daughter Auction can bid here, without having to pay higher grid fees. Unfortunately, this function is not fully implemented, meaning that at the moment, its functionality is the same as the Daughter Auction.
Bilateral Trading Figure A.5	The Bilateral Trading smart contract offers an implementation for bilateral trading between two market participants. Participants can register new offers together with the amount of electricity, the minimum/maximum price at which they are willing to sell/buy, and the bidding time (the time period after which the offer will end). Consequently, the smart contract also offers the possibility to send bids for open offers, and if the new bid is better than the old one, the sender of the new bid becomes stored as the highest bidder. Bids are always directed to a specific address and only contain the price that the bidder is willing to pay/sell for. No function is implemented that would execute the expired offers.
Dynamic Grid Fee Figure A.6	The idea behind this is to charge grid fees based on the load of the trafo or the circuit, but the current implementation is not finished.

Table 3.1 – Oli Smart Contracts [54]

The market implementation is fairly simple, but shows some flaws. Many functions are not finished and the market's entirety depends on the calling of essential market functions, such as the breakEven and reset functions, from outside. Because of this, the advantages of implementing such a market on a blockchain are debatable. Additionally, the market must rely on the forecast accuracy of the bids. In the current

implementation, a 100% accuracy is assumed. However, the ideas presented by Oli Systems are a good foundation for the market implementation of this thesis.

3.3.2 Tobalaba / Energy Web Foundation

"The Energy Web is an open-source, scalable blockchain platform specifically designed for the energy sector's regulatory, operational, and market needs. It serves as a foundational, shared, digital infrastructure for the energy and blockchain community to build and run their solutions. Together, the EWF, Affiliates, and Community are unleashing blockchain's potential to accelerate the transition to a decentralized, democratized, decarbonized, and resilient energy system." [55]

On November 1st, 2017, the EWF granted access to its test network, called Tobalaba. Tobalaba is Ethereum-based and uses Aura, a PoA algorithm that organizes the signing of blocks in rounds. Every validator or authority node receives a time slot per round in which one new block can be signed. In the case of the Tobalaba test network, these time slots are about 3 seconds long. If the current validator is not responding within the given time, it will be skipped and the validator that is next in the order will have the chance to sign one new block. [56]

At the time of this thesis, Tobalaba was being tested in its beta release phase. Developers and energy companies could request test tokens and create an account on Tobalaba. The EWF also offered a netstats page (http://netstats.energyweb.org/) listing the most important current network stats. The Gas Limit per block was 80,000,000 [55], which is 10 times higher than the Gas Limit on Ethereum's public chain, and the average block time was about 3.9 seconds [57] (roughly a third of the public Ethereum network). The higher Gas Limit and shorter block time allowed Tobalaba to provide a throughput of up to 750 TPS in comparison to Ethereum's 15-20 TPS. Additionally, Polkadot (implementation of subchains) and payment channels are planned to be implemented in the future, which will secure a much higher throughput.

This section shows the growing interest of scientists and entrepreneurs for blockchain in the energy sector. Most of the implementations are fairly simple and exist as prototypes with little access to detailed information. Oli Systems delivered a more detailed overview of their very simple market implementation, which contributed a great deal to this thesis, as many ideas of their market are adopted and improved in it. As for the trading periods of the market implementations, most are too long to be feasible for real-life implementation, especially if ancillary services are to be offered. Additionally, little focus was given on the blockchain performance and its scalability.

Chapter 4

Concept

In this thesis, different LEMs are designed and implemented. The core architecture always stays the same, but certain grid-stabilizing measures can be thought of as completely decentralized (where the LEM tries to achieve grid stability without monetary help from outside) or semi autark (where outside regulators offer money for grid-stabilizing measures). These concepts are explained in more detail later in this chapter.

The following Figure 4.1 shows an overview of the blockchain-based LEM that is implemented. The On-Chain and Off-Chain differentiation says whether the features are implemented on the blockchain, or take form of additional program code that only communicates with the blockchain:

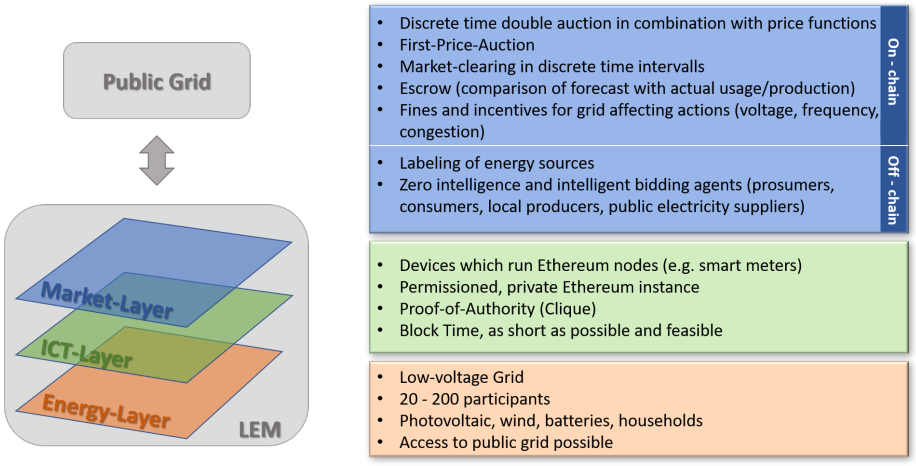


Figure 4.1 – Concept for LEM

On the market layer, a first price DTDA is implemented in combination with price functions for voltage, frequency, and congestion. The market clearing happens

in discrete time periods, aimed to be made as short as possible in order to offer grid-stabilizing measures as fast as possible. The bids at the DTDA are based on forecasts sent from ZI and intelligent bidding agents. After the bids are sent and the market is cleared, the agents accurately submit the amount they used and the Escrow function finalizes the transactions.

The information layer consists of devices that are able to run Ethereum nodes (through geth) that are connected to a permissioned and private instance of Ethereum. The consensus mechanism used is clique. The most feasible block time is tested later in this thesis. Figure 4.2 illustrates the information layer. On the network level, every node is connected to a least one other node. However, not all nodes have to be directly connected to every other node in order to be able to synchronize with the whole network/blockchain.

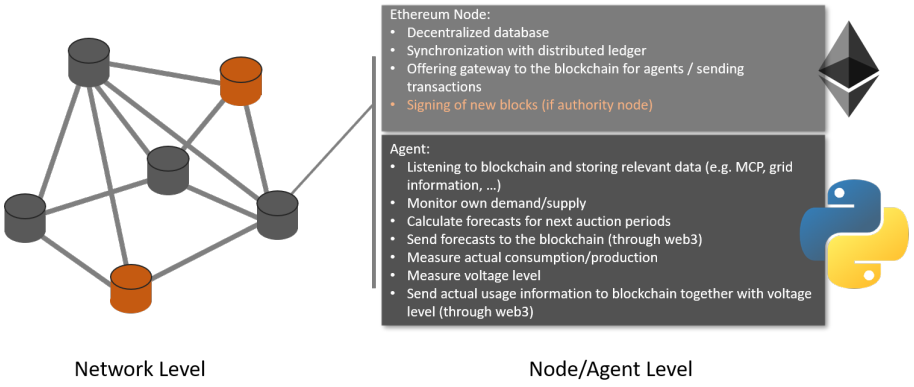


Figure 4.2 – Information Layer Illustration

The simulated energy layer consists of 20 to 200 participants who are connected within a low-voltage grid and enabled to trade electricity with each other. Participants either buy (households, batteries) or sell power (photovoltaic, wind, batteries). In addition to the LEM, participants can also buy and sell power to the public grid.

4.1 Assumptions

Before we can go into more detail about the concept behind LEM, a number of assumptions must be defined:

- At the LEM, only electricity is tradeable.
- All participants are connected via an electricity grid.
- Electricity trading between participants is assumed to be "behind-the-meter", meaning no regulations take effect.

- The local electricity grid is connected to the normal grid. Electricity can be drawn and fed-in at fixed prices.
- Every agent that wants to connect to the market must first be approved by a consortium.
- Every agent is assumed to have a smart meter with a built-in private key.
- Every connected agent is assumed to measure and report its usage or production in a correct and honest manner.
- Every agent that has been approved is able to connect to the blockchain and has enough computing power to run a light blockchain client.
- The underlying cryptocurrency is directly linked to a fiat currency, e.g. Euro.

4.2 Market Design

In this section, the market design is explained in more detail. First, some general information is given about the market, then the used auction mechanism is explained, followed by the price functions that are implemented in order to ensure the grid's stability, and lastly, the different approaches that can be chosen.

4.2.1 General Information

The market is based on all of the assumptions mentioned in Section 4.1. Every participant is an agent with its own account on the blockchain. These accounts transfer FAUTokens, which are a cryptocurrency in which 1000 FAUTokens equals the value of 1 Euro.

4.2.2 Auction Mechanism

The implemented auction mechanism is a DTDA. The DTDA enables the participants to trade energy based on their current demand or supply and the clearing price reacts to energy surpluses or deficits. This way, participants are more likely to invest in energy storage (batteries) and adapt their demand according to the supply, which will eventually flatten the supply/demand curves.

The participants of this auction are all connected to the same trafo but can be connected to different circuits. The following figure Figure 4.3 shows an example of a LEM with different circuits and one trafo.

Participants send bids to the auction based on a forecast of how much energy they will consume or demand in the upcoming auction period. In addition to the

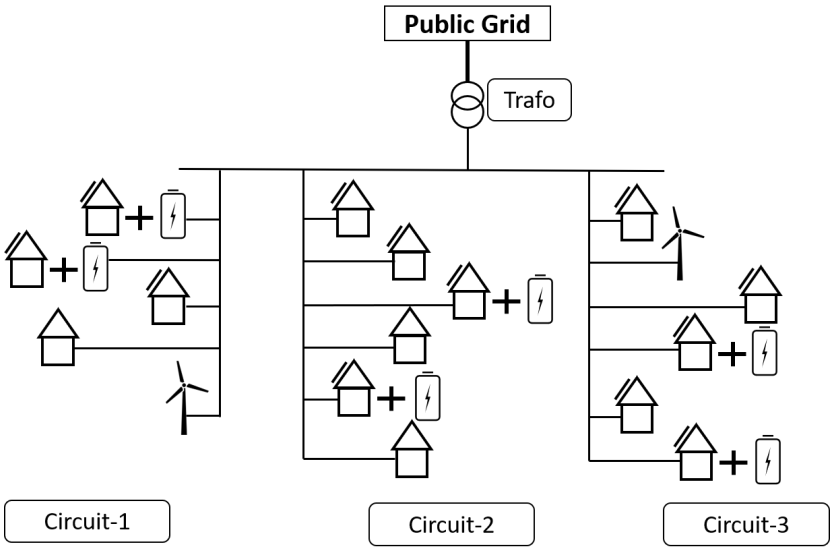


Figure 4.3 – Example Grid

amount and price of their bid or ask, they include the purpose of the energy demand, such as households or industry, or the origin of the energy supply, such as wind or photovoltaic. The auction has a fixed period after which the market clears and the MCP is identified. After the MCP has been identified, the agents measure their consumption and production in the following auction period. After this period is enclosed, participants send information containing the actual amount that they have consumed or produced during the next auction period. Furthermore, they include the voltage level measured at their node. After another auction period is over, it is assumed that all participants sent the information of their actual supply/demand, and the Escrow function is executed. Figure 4.4 visualizes the order of events.

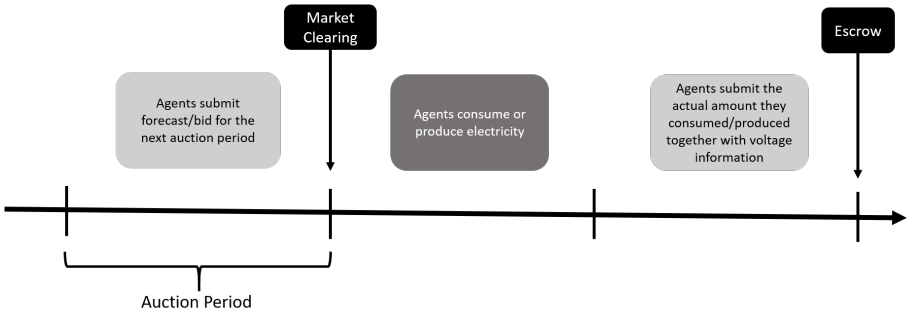


Figure 4.4 – Order of Events

The Escrow function determines how much electricity can be traded within the market and how much must be procured from the public grid. Afterwards, it assigns

which participants are able to trade within the market. All producers that offered energy at either the same or a lower rate than the MCP are allowed to trade within the market for the MCP. Similarly, all consumers that offered energy at either the same or a higher rate than the MCP are allowed to trade within the market for the MCP. The successful participants have to trade the same amount of energy as their forecast (the bidding amount), because the MCP is calculated based on that. The difference between the forecast and the actual amount has to be allocated from the public grid. This way agents are also more likely to send accurate forecasts because every interaction with the public grid can be seen as negative since it offers worse prices for the market participants and the possibility of additional congestion fees exists. Participants that were not successful at the auction have to transact their energy with the public grid. The energy that is allocated within the market is traded at the MCP and the energy that is allocated from the public grid is traded either at the public grids feed-in tariffs or the public grid consumption price. In addition to the MCP, participants may also have to pay fees for grid-stabilizing measures. These are explained in more detail in section Section 4.2.3.

The following example explains the auction mechanism in more detail. Participant-1 sends a forecast saying that 670 Wh will be produced at 11.3 cents within the next auction period. The market then clears and its clearing price is 12.2 cents. This means that participant-1 was successful at the auction. After the auction period is over, participant-1 measures the actual amount produced, which ends up being only 640 Wh. Because of this success at the auction, the participant's forecast amount now has to be traded within the market at the MCP. Hence, the missing amount (30 Wh) is now required from the public grid.

4.2.3 Grid Fees

One of the main focuses of this thesis is the grid-friendliness of the implemented energy market. As explained in Chapter 2, many factors influence the grid and many parties participate in order to keep the grid stable. This market is stabilized by price functions for the main influences on the grid: congestion, frequency, and voltage.

Figure 4.5 shows where the different fees apply in the grid. The frequency remains constant for the whole grid, congestion is measured per trafo, and voltage is measured per circuit. The following explains the price functions for these three influences in more detail.

Congestion

Congestion is measured per trafo. Based on the information of actual usage, which is sent after the auction period in which the agents actually consume/produce the energy, the difference between the total amount consumed and the total amount

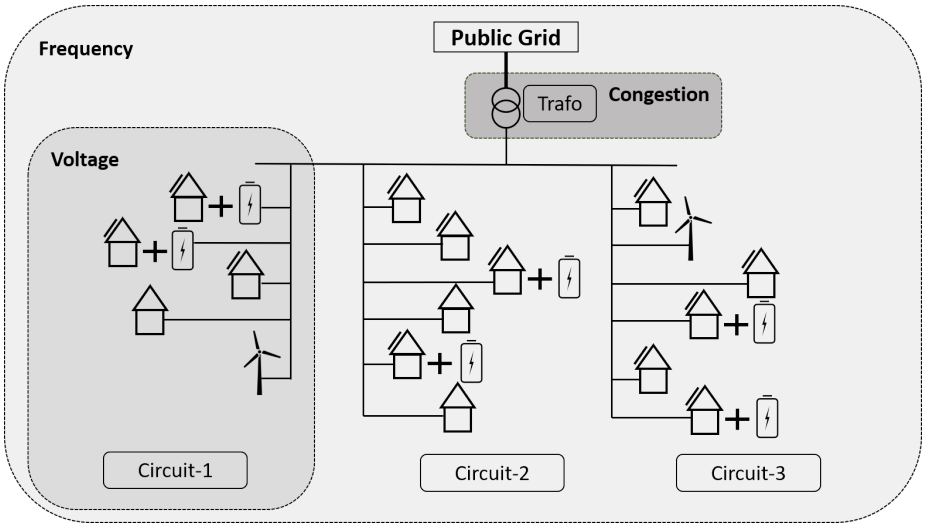


Figure 4.5 – Example Grid with Grid Fees

produced is calculated. The surplus or deficit is the amount of energy which has to flow through the trafo, out of or into the LEM. Depending on the trafos quality, size, and other characteristics, a maximum load, which the trafo is able to handle, is defined. Overshooting this maximum load would lead to a congestion issue which can, by extension, lead to a break down of the entire trafo/transmission line.

Multiple fee-levels are implemented based on the maximum load. If the trafo operates lower than 50% of the maximum load, no congestion fees are added. If it operates between 50 and 80% of the maximum load, a small fee is charged for every kWh that is transferred over the trafo. If the load is higher than 80%, a higher fee is charged for every kWh that is transferred over the trafo. Only participants that raise the deficit or surplus (e.g. the LEM has a surplus, leading a producer to thus raise the surplus) get charged the fee. Participants that produce a counterflow (e.g. consumers during times of a surplus) receive money based on the fee-level because they support the grid's stability.

Frequency

Frequency remains the same at every node in the entire grid. For this reason, only the trafo needs to measure the frequency and submit the information once for every auction period. Again, three fee-levels are implemented which depend on the deviation of the frequency from the standard frequency of 50Hz. A deviation of less than 0.08Hz (in any direction) leads to no additional charges. A deviation between 0.08Hz and 0.14Hz leads to a small extra fee per kWh. A deviation which is higher than 0.14Hz leads to a higher extra fee per kWh. Again, only participants that act against the stability of the grid (e.g. produce energy during times of high

frequency) are penalized. Participants that act supporting for the stability of the grid (e.g. consume energy during times of high frequency) receive money for their grid-stabilizing measures. At short auction period times of less than 30 seconds, this type of measure can be seen as FRR, as explained in Section 2.1.2.

Voltage

The voltage level is measured at every node. Each node submits its voltage level and in the escrow function, each circuit identifies its critical node (the node with the highest deviation from the standard voltage of 230V). Again, three fee-levels are implemented which depend on the deviation from the standard voltage of 230V. A deviation of up to 5V leads to a small extra charge. A deviation of up to 10V leads to a medium extra charge. Everything above that results in a high extra charge. Only grid-unfriendly measures are penalized (e.g. feeding in energy during times of high voltage), while grid-friendly measures are supported (e.g. consuming during time of high voltage).

4.2.4 Different Approaches

This thesis aims to design different markets with different approaches. That is why all of the aforementioned price functions in Section 4.2.3 are modular and can either be added to the market, or not, depending on the use case and simulation/analysis reasons.

The price functions in Section 4.2.3 all explain the semi-autark approach of achieving grid-friendly behavior. It is not completely autark because the amount of money that is spent on grid-friendly behavior and the amount of money spent on non-grid-friendly behavior are not equal. Grid-friendly and non-grid-friendly measures receive the same monetary value per Wh. Consequently, a money pool is necessary, where money is collected (from grid-unfriendly participants) and spent (to grid-friendly participants). For this approach, the money pool is just assumed to be existent (e.g. collected through grid usage fees, paid by the DSO, etc.).

Another approach is the fully-autark approach. Instead of charging and spending the same amount of money for grid-friendly and non-grid-friendly actions, this approach only collects money for non-grid-friendly measures based on the corresponding fee-level. This amount of money is then solely used to pay the grid-friendly participants. Hence, grid-friendly participants do not receive money based on the fee-levels, but rather based on the amount of money that has been collected from the non-grid-friendly participants. This way, no money pool which is funded by outside parties or additional usage fees are necessary.

Both of the previously mentioned approaches are implemented for frequency and voltage, and can be chosen on contract deployment.

4.3 Agents

Agents are, as explained in Section 3.2, software programs that act as representatives of market participants. In this thesis, an agent can represent either producers of energy (e.g. photovoltaic or wind) or consumers of energy (e.g. households that can have their own photovoltaic modules installed and different sizes of batteries). Other energy suppliers or consumers can also be represented by agents, but are not implemented.

Every agent becomes registered at the market with a trafo and circuit number. It receives a unique account address for token transfers and identification. During registration, the account will get pre-loaded with sufficient funds for the first few trades. The intelligent bidding agent aims to be cost-efficient, which means that it wants to buy energy cheap, but sell it as expensive as possible. It bids based on the current MCP, its battery load, and the grid's voltage. In addition, different bidding types are implemented that allow more conservative (higher chance to trade within the LEM, but at a lower profit) or more aggressive (lower chance to trade within the LEM, but at a higher profit) bidding. The ZI agent simply sends random bids within a predefined price range.

Agents forecast their amount for the next auction period as well as possible because the higher the deviation from the forecast, the more the agent has to trade with the public grid, which is more expensive than trading within the LEM. The actual usage amounts are reported accurately by the agents. In order to be able to interact at the DTDA market with other agents, all agents have to be connected to the same trafo and must have an account containing a defined minimum balance in FAUTokens. They can submit one bid and one actual usage per auction period. The actual usage can only be submitted if they send a bid during the corresponding auction period.

4.4 Blockchain

For the information layer, the blockchain technology was chosen over a common database. It offers many benefits, such as higher security, higher transparency, no intermediaries, and its decentralized nature fits the decentralization of energy production. Moreover, it is a fairly new technology that still needs to be tested and reviewed in order to obtain more information for further improvements.

In more detail - a private, permissioned instance of the Ethereum blockchain is used. Ethereum is open-source, it has by far the largest development environment for dApps, and the largest developer community standing behind it. This makes it the perfect blockchain for the implementation of a prototype. In terms of a real-life

implementation, on the other hand, other blockchains, such as Hyperledger Fabric, are possibly a better fit. This is especially probable due to its abilities to restrict view-ability and assign different roles.

Since participants who want to join the LEM must first be authorized and auction period times should be as constant as possible, PoA is the ideal consensus mechanism for this use case. In addition, no electricity is wasted on PoW and more TPS are possible. For the PoA mechanism, either a handful of validators can be assigned or every participant can be a validator. The more validators, the more secure and fair the network is, but it also makes things more complex. In a real-life implementation it could be possible to have all participants vote for validators in a pre-defined time interval.

The consensus mechanism used is clique. Clique is a PoA consensus mechanism that validates blocks in rounds. After each block period (or block time), a new block is signed by the authority node that is in-turn. Authority nodes can either propose the creation of new authority nodes or propose to degrade other authority nodes. Once more than 50% of the signers come to a consensus, the measure takes place. Another PoA algorithm is Aura, which is available in Parity, an Ethereum client platform. Aura also signs blocks in rounds, based on the current timestamp. However, for this thesis we decided on using geth as our Ethereum client platform and, consequently, clique is used as PoA consensus mechanism.

Chapter 5

Implementation

In this chapter the actual on-chain implementation of the LEM in Solidity is described along with the agents that are necessary for the off-chain interaction with the smart contracts. Before this description, however, a little introduction is given about the development environment and blockchains settings.

5.1 Development Environment

The main programs, libraries, and programming languages that were used for this thesis are briefly explained in this chapter.

The python modules listed in Table 5.1 were used for the implementation of the python scripts.

Module Name	Module Version
csv	1.0
json	2.0.9
logging	0.5.1.2
pandas	0.23.3
psutil	5.4.7
solc	0.4.23
web3	4.2.0

Table 5.1 – Used Python Modules

5.1.1 Solidity

Solidity is Ethereum's most common programming language. The primary development of Solidity was done by Gavin Wood, Christian Reitwiessner, and several other Ethereum core contributors [58]. It is a contract-oriented, high-level language

for implementing smart contracts, which are then compiled to bytecode and can be executed on the EVM. Its syntax was designed around the ECMASript syntax which is also the foundation of JavaScript. This is what makes Solidity primarily familiar to web developers, but it was also influenced by C++ and Python [59].

Aside from Solidity, the programming languages Serpent (similar to Python, but deprecated), LLL (low-level-Lisp like languages), Viper (Python-derived, but so far only experimental), and Mutan(Go-based, but deprecated) were also designed to target the EVM. [60]

Solidity is a very simple programming language. In order to use pre-defined functions, the libraries that include these functions have to first be deployed as smart contracts. All variables have to be initialized in the correct type and size, and for some mathematical operations they must often be converted explicitly. One of the major drawbacks, however, is that Solidity doesn't yet allow floating point numbers, making calculations oftentimes difficult. Smart Contracts in Solidity also have a bytecode cap, which means that a programming code that is too long cannot be deployed on the blockchain and has to be wrapped in multiple, smaller smart contracts.

5.1.2 Remix IDE

Remix is Solidities most popular Integrated Development Environment (IDE). It is open source and allows for the writing of smart contracts straight from the browser, but can also be used locally. It is a very powerful tool for testing, debugging, and deploying of smart contracts. It has a built-in compiler and testing environment which allows the user to choose to use either the built-in JavaScript VM as blockchain simulation, or connect to a running blockchain via Web3. [61]

Even though remix is the most popular Ethereum IDE, it still comes with many flaws. For example, the compiler doesn't catch missing return statements or other common programming mistakes (e.g. the use of only one '=' instead of '==' for the comparison of two variables).

5.1.3 Web3.py

Web3.py is a python library derived from Web3.js [62]; a Javascript API for interacting with local or remote Ethereum nodes through an HTTP or IPC connection [63]. Both offer all necessary functions for deploying smart contracts, sending transactions, processing transaction receipts, and reading general information about the blockchains state, such as blocknumber, gas limit, etc.

5.1.4 Ganache

Ganache is a tool for simulating a private Ethereum test-blockchain. It is used for the deployment of smart contracts and the testing of interaction with them. Formerly known as TestRPC, which was only available as a command-line tool, Ganache is also available as a desktop application. The user has the ability to create as many accounts as desired, which can all be pre-funded. The Graphical User Interface (GUI) shows up under the tab "Accounts", listing all accounts with their address, current balance, and the number of transactions they have sent so far. The tab "Blocks" shows all blocks that have been mined, along with the gas used and the transactions made. The tab "Transactions" lists all transactions and any additional information corresponding to them. Under the tab "Logs", all server logs are stored. Every time Ganache is initiated, the blockchain can be configured as desired. The gas limit and gas price can be adjusted, along with the network id and port number. Additionally, the user can choose if the miner should mine every new transaction instantly ("automine") or simply mine after a pre-set block time. [64]

Unfortunately, the parameter "gasUsed", located under the "Blocks" tab doesn't show the gas used per transaction, but rather the cumulated gas used so far in this block. This is a known problem that hinders the testing of sending transactions. Another challenge is that Ganache can only handle a small number of connections at the same time and throws read and socket timeouts whenever too many accounts try to send transactions simultaneously. In addition, it seems to have challenges with calculation-intensive transactions (with a high gas usage). These can lead to block times that are longer than the predefined block time.

5.1.5 Geth

Geth is the official Go implementation of the Ethereum protocol [65]. It offers a command line interface for running a full Ethereum node. The node can either connect to an existing Ethereum network or be part of a new private Ethereum network. Geth nodes can mine ether, transfer funds, create contracts, send transactions, explore block history and much more [66]. The interaction with geth happens through a javascript console that includes the web3.js dApp Application Programming Interface (API).

Geth is an easy way of setting up one's own Ethereum node and it offers many options for node synchronization.

5.2 Blockchain Setting

For simulation purposes, we use not only Ganache, but also a real blockchain consisting of 20 Virtual Machines (VMs), which will be referenced as i7Chain in this thesis. Ganache is a great tool for testing and creating short-time simulations because it is easy to set up and manage. However, when it comes to longer simulation times and more socket connections, Ganaches performance worsens, becoming buggy.

The i7Chain is more reliable and closer to a real-life implementation. Therefore, Ganache is only used for testing purposes in this thesis, while the i7Chain is used for actual simulations and the analysis of the market, as well as blockchain performance.

For the i7Chain a server with two Intel Xeon E5-2637 v4 @ 3.50 GHz Central Processing Units (CPUs) and 80Gb of RAM is used. This way, every node of the blockchain, or every VM, has approximately 350 MHz of computing power and 4096 Mb RAM. All VMs run on Ubuntu 18.04 and have geth (version 1.8.14), python3, and some additional python3 libraries installed.

The block time is set to 5 seconds so that each auction period (which is set to 30 seconds) consists of six blocks on average. This way, participants have a very low chance of missing to submit a bid to the current auction period. In addition, the negative impact of forks is, for the most part, prevented, since forks with a length of more than six blocks are very uncommon with this consensus mechanism. Tests have shown that 3 to 5 seconds is the smallest block time that is possible in order to ensure a stable blockchain where all nodes are synchronized properly. The auction period is set to 30 seconds because we wanted it to be as short as possible to offer grid-stabilizing measures as fast as possible, but we also want to prevent forks and minimize the risk for participants to miss their bidding windows. The numbers mentioned are only valid for a chain with about 20 nodes. The details are explained in Section 6.2.4. Additionally, short block and auction period times lead to more disk usage. This correlation is analyzed later in Section 6.2.3. The gas limit per block is set to 20,000,000, which is also analyzed in more detail later.

The i7Chain is initialized by the genesis file shown in Listing 5.1. The config parameter in Listing 5.1 holds the blockchain configuration. ChainID is the network identifier; it is an arbitrary value that will be used to pair all nodes of the same network. This value must be different from 0 to 3 because these are already used by the live chains (mainnet chainID is 1). The homesteadBlock parameter defines whether you are using the Homestead release (value 0), which is the second major release of Ethereum, or if you are using Frontier (value 1), which is the first major release of Ethereum. The parameters that start with "eip" stand for Ethereum Improvement Protocol, where developers propose how to improve Ethereum. These proposals can be accepted and used for one's own chain or not. The eip150Hash is necessary for the fast sync mode of the geth client. [67]

```
{
"config": {
"chainId": 222,
"homesteadBlock": 1,
"eip150Block": 2,
"eip150Hash":
"0x0000000000000000000000000000000000000000000000000000000000000000
0000000",
"eip155Block": 3,
"eip158Block": 3,
"byzantiumBlock": 4,
"clique": {
"period": 5,
"epoch": 30000
}
},
"nonce": "0x0",
"timestamp": "0x5a0eb4aa",
"extraData":
"0x446f206f7220646f206e6f74202d207468657265206973206e6f20747279
0000ef598289403499a829235957350fbf29de57f473000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000",
"gasLimit": "0xffffff",
"difficulty": "0x1",
"mixHash":
"0x0000000000000000000000000000000000000000000000000000000000000000
0000",
"coinbase": "0x0000000000000000000000000000000000000000",
"alloc": {
"ef598289403499a829235957350fbf29de57f473": {
"balance":
"0x2000000000000000000000000000000000000000000000000000000000000000
0000000"
}

...

"07ad03d56ae65e4b57322263011cbad6c1d6b9fd": {
"balance":
"0x2000000000000000000000000000000000000000000000000000000000000000
0000000"
}
},
"number": "0x0",
"gasUsed": "0x0",
"parentHash":
"0x0000000000000000000000000000000000000000000000000000000000000000
0000"
}
```

Listing 5.1 – genesis.json

A very important parameter for this thesis is clique. It is the used PoA consensus mechanism. The period parameter in clique defines the block time (or the minimum difference between two consecutive block's timestamps). For this implementation, it is a duration of 5 seconds. The epoch parameter defines the number of blocks after which the open proposal votes are reset. [68]

"Mixhash and nonce are used together to determine if the block was mined properly. The reason we have both is that if an attacker forges blocks with a false nonce, it can still be computationally costly for other nodes in the network to discover that the nonce was falsified. Mixhash is an intermediary calculation to finding the nonce that is not as costly to determine. Thus, if other nodes on the network discover an errant mixhash when validating a block, they can discard the block without doing additional work to checking the nonce." [67]

This "intermediary calculation" is done by generating only a small amount of the Directed Acyclic Graphs (DAG) instead of using the entire DAG for verification. The DAG can be seen as a vast dataset and is only necessary for PoW mechanisms.

Timestamp defines when the block was created in unix values. The extraData parameter holds information on the signers, such as arbitrary signer vanity data and, in the case of the usage of PoA as a consensus mechanism, the addresses of nodes that are allowed to sign [68]. The gasLimit parameter defines the maximum amount of gas that can be spent in the genesis block. For this blockchain we decided to use 16.7 million gas, which is about two times the gas limit of the Ethereum mainnet. As explained in Section 2.2.3, the gasLimit is not constant in Ethereum and can change after each block, depending on the signers' information. The difficulty parameter is not important for a PoA chain, it is only important for PoW chains, since it determines how hard it is to mine a block. The coinbase parameter defines where the reward for mining the genesis block goes. In the alloc parameter exists a list of all addresses that are participating at the blockchain from the beginning. In addition, their starting balances in Ether can be defined here. The parentHash parameter is meaningless since the genesis block has no parent. However, the goal was to make the genesis block as similar to all other blocks of the chain as possible. [67]

Once the blockchain has been initialized with the genesis file, the signers may begin validating the blocks. In order to synchronize the network, the starting signer node has to know the enode information of the other nodes that are connected to the chain. This can be achieved through listing the enode information of all nodes in the static-nodes.json file.

In our setup, we usually use only one signer in order to prevent forks, but multiple miners can be added if they are proposed through the clique consensus mechanism.

5.3 Market

The entire functionality of the DTDA market is implemented in solidity smart contracts. The solidity version 0.4.23 is used. The following sections explain the various smart contracts necessary in making the whole market work.

5.3.1 FAUCoin

The FAUCoin smart contract introduces a new token, the FAUToken, which is assumed to be directly linked to the Euro. 1000 FAUTokens equal 1 Euro. This contract enables transactions of this token between accounts. The following Figure 5.1 shows attributes and methods of the FAUCoin smart contract.

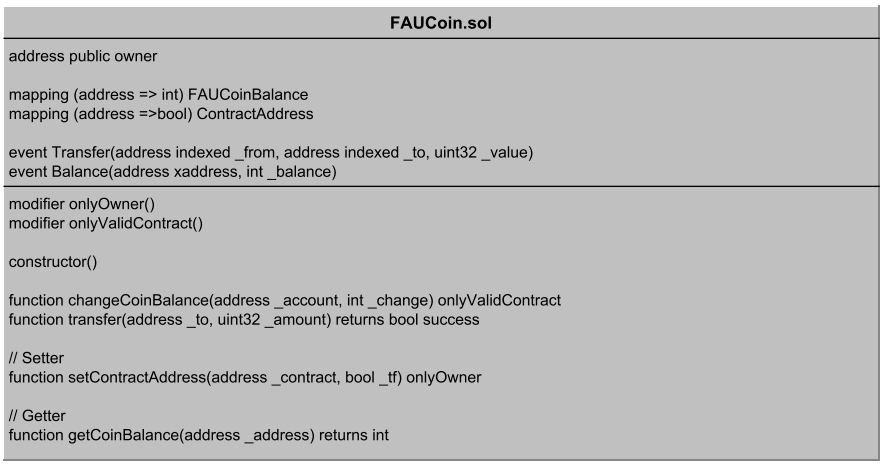


Figure 5.1 – i7Chain Smart Contracts FAUCoin.sol

The owner variable stores the address of the account that deployed the smart contract. It is set during deployment in the constructor. The FAUCoinBalance mapping stores balance information regarding the different addresses that are participating in the market. The ContractAddress mapping stores information stating whether a contract or address has certain privileges. Usually, the DoubleAuction contract is set to true here in order to be able to use functions that are restricted by the onlyValidContract modifier. Since it is possible to have multiple DoubleAuction contracts for multiple LEMs, this information is stored in a mapping.

The Transfer and Balance events simply emit information. Transfer emits the two addresses that participated in the transaction and the amount. Balance returns the balance of a specific address at the end of the changeCoinBalance function.

The onlyOwner modifier restricts functions so that only the FAUCoin contract owner can call them, and the onlyValidContract modifier restricts functions so that

only addresses that are set true in the ContractAddress mapping are allowed to call these functions. The constructor is called during deployment of the contract and sets the owner value to the messenger sender address.

The changeCoinBalance function is only callable by addresses that are set to true in the ContractAddress mapping. It takes the old balance of the passed-on address from the FAUCoinBalance mapping, adds the value of the passed-on integer to it, and stores the new value in the FAUCoinBalance mapping. Afterwards, the Balance event is emitted.

The transfer function needs a receiver address and an amount as parameters. It first checks if the message sender has sufficient funds on his account (more than the value that he is trying to transfer) and then transfers the money. The account balance of the message sender decreases in FAUCoinBalance by the transferred amount, while the account balance of the receiver address increases by the same amount. In addition, the Transfer event is emitted and true is returned.

The setContractAddress function simply sets the address submitted in the parameters to true in the ContractAddress mapping. This function can only be called by the owner of the FAUCoin contract.

The getCoinBalance function looks up the submitted address value in the FAUCoinBalance mapping and returns it. For testing purposes, this function does not have any modifiers, but in a real-life implementation, the call of the function should be more restricted in order to make it more difficult for outsiders to retrieve information about other account balances.

5.3.2 Register

The Register smart contract is necessary for registering new participants, as well as trafos, at the market. All market participants have to be registered here first before they can start trading at a double auction market. Figure 5.2 shows the class diagram of the Register smart contract.

The FAUCoin coin variable is a reference to an FAUCoin smart contract. The owner variable stores the address of the contract deployer. The AddressMapping mapping stores all relevant information for each participant in the form of a Details struct, which holds information regarding both the trafoID and circuit the participant is connected to, as well as the label information, which is an integer value that can later be translated into household, photovoltaic, wind, etc. A complete list of the different energy types is shown in Table 5.2. The TrafoMapping mapping is similar, but its struct holds different information. It stores the trafoID, the maximum load of the trafo, and three account addresses for congestion, frequency, and voltage fees. The Trafos mapping maps the trafoIDs to the addresses of the specific trafo.

Register.sol

```
FAUCoin coin

address public owner

mapping (address => Details) AddressMapping
mapping (address => DetailsTrafo) TrafoMapping
mapping (uint32 => address) Trafos

event NewRegistered(address paymentAddress)
event TrafoBalances(int BalanceGrid, int BalanceCongestion, int BalanceFrequency, int BalanceVoltage, uint256 AuctionID)

struct Details {uint32 trafo; uint8 circuit; uint8 label; }
struct DetailsTrafo { address addrCongestion; address addrFrequency; address addrVoltage; uint32 trafo;uint32 maxLoad; }

modifier onlyOwner
modifier onlyAddressOwner

constructor()

function addParticipant(address _address, uint32 _trafo, uint8 _circuit, uint8 _label) onlyOwner
function addTrafo (address _addressGrid,address _addressCong,address _addressFreq,address _addressVolt,
... uint32 _trafo, uint32 _maxLoad) onlyOwner

// Setter
function setFAUCoin(address addr) onlyOwner
function setLabel(address _address, uint8 _label) onlyAddressOwner

// Getter
function getLabel(address _account)  returns uint8
function getCircuit(address _account) returns uint8
function getTrafoID(address _account) returns uint32
function getTrafoAddr(uint32 _tid) returns address
function getTrafoLoad(uint32 _tid) returns uint32
function getTrafoCongestionAddr(uint32 _tid) returns address
function getTrafoFrequencyAddr(uint32 _tid) returns address
function getTrafoVoltageAddr(uint32 _tid) returns address
```

Figure 5.2 – i7Chain Smart Contracts Register.sol

Energy Label	Energy Type
1	Biomass
2	Solar Power
3	Battery (sells to market)
4	Wind Power
5	Coal / Gas / Oil
6	Combined Heat and Power System
7	Hydro Power
8	USED BY CODE
11	Household Load
12	Industry Loads
13	Battery (buys from market)
18	USED BY CODE

Table 5.2 – Energy Type Encryption

Furthermore, the contract has two events implemented. The NewRegistered event, which emits the address of a newly registered participant, and the TrafoBalances event, which returns all relevant balances of a trafo together with its corresponding AuctionID.

The modifier onlyOwner requires that the message sender of a transaction be the same as the deployer of the smart contract. The modifier onlyAddressOwner requires the message sender to have the same address as that which was used as the passed-on parameter in the function itself. In the constructor, only the owner variable is set.

The function addParticipant adds a new participant to the contract. Necessary information includes the participant's address, the trafoID, and the circuit to which it is connected to, as well as its type of production or consumption. All of this information is stored in the AddressMapping mapping in the form of a struct. In addition, the newly registered address will be pre-funded with sufficient funds for trading (for testing purposes) by calling the changeCoinBalance function of the coin object. Ultimately, the NewRegistered event is emitted.

The addTrafo function adds a new trafo to the contract. It needs four addresses as parameters for the grid fees and public grid. In addition, the trafoID and the maximum load is passed-on to this function. All of the information becomes mapped in the form of a struct to the corresponding public grid address in the TrafoMapping mapping, and the public grid address becomes mapped to the trafoID in the Trafos mapping. Afterwards, all four trafo addresses are pre-funded with sufficient funds (for testing purposes).

The setFAUCoin function simply sets the coin variable of the contract to the passed-on address of an FAUCoin contract, and can only be executed by the contract owner. The setLabel function changes the label of a participant in the AddressMapping mapping. This action is only allowed by the address owner.

Since this contract holds a great deal of important information, it also has many getter functions. The getLabel function returns the label based on a participant's address. Similarly, the getCircuit function returns the circuit, and the getTrafoID returns the trafoID, both based on a participant's address. The getTrafoAdr function returns the address of a trafo based on its trafoID. The functions getTrafoLoad, getTrafoCongestionAddr, getTrafoFrequencyAddr, and getTrafoVoltageAddr return the corresponding struct values based on the trafos trafoID.

5.3.3 Double Auction

The DoubleAuction smart contract is, essentially, the market's core contract. It is necessary for bidding, clearing the market, collecting the actual usage, and finalizing the auction. Figure 5.3 shows the class diagram of the DoubleAuction contract.

DoubleAuction.sol

```
address public owner

Register Reg
FAUCoin Coin
GridFee Fee

uint AuctionID
uint32 Trafo
uint16 GridPriceBuy
uint16 GridPriceSell
uint16 MaxRate = 0
uint16 MinRate = 65000
uint16 AuctionTime = 30

bool constant ModCongestionFee = true
bool constant ModAutarkFrequencyFee = true
bool constant ModSemiAutarkFrequencyFee = true
bool constant ModAutarkVoltageFee = true
bool constant ModSemiAutarkVoltageFee = true

mapping (uint256 => Details[]) ParticipantsBids
mapping (uint256 => Details[]) ParticipantsUsage
mapping (uint256 => mapping (address => bool)) BidderBool
mapping (uint256 => mapping (address => bool)) UsageBool
mapping (uint256 => uint256) Frequencies
mapping (uint256 => uint64) MCPs
mapping (uint256 => uint256) Timer

uint32[] SRate = new uint32[](300)
uint32[] DRate = new uint32[](300)

event NewBid(address indexed addr, uint64 rate, uint32 amount, uint8 label, uint256 CurrentAuctionID, uint64 LastMCP)
event NewMcp(uint64 cbid, uint16 minRate, uint16 maxRate, uint256 NewBlockTime, uint256 AuctionID)
event NewUsage(uint64 _price, uint32 _amount, uint8 _label, uint32 voltage, uint256 _AuctionID, address xaddress)
event EscrowEvent (uint8 flag, uint32 amountProduced, uint32 amountConsumed, uint256 numberParticipants,
... uint256 _AuctionID)
event Transfer(address xaddress, uint32 _amount, uint64 _TransferPrice, int reward, int FrequencyFee, int VoltageFee,
... int64 CongestionFee, uint8 TransferType, int Balance)
event TimerEvent(string output, uint256 LastBlockTime, uint256 CurrentBlockTime)

struct Details {uint64 price; uint32 amount; uint8 label; address xaddress; uint32 amount2; }

constructor()

modifier onlyOwner()
modifier onlyValidProsumers()
modifier onlyValidBidders()
modifier onlyOneBidPerPeriod()
modifier onlyOneUsagePerBlock()

function checkLabel(uint8 _label) returns uint8
function preBid(uint32 _amount, uint16 _price, uint8 _label) onlyValidProsumers
function bid(uint32 _amount, uint16 _price, uint8 _label) onlyOneBidPerPeriod
function marketClear() returns uint64
function reset()
function actualUsage(uint32 _voltage, uint32 _amount, uint8 _label, uint256 _AuctionID) onlyValidBidders
... onlyOneUsagePerBlock
function innerTransferMoney(uint256 _AuctionID, uint32 _amount, uint8 _label, address _address)
function outerTransferMoney(uint256 _AuctionID, uint32 _amount, uint8 _label, address _address)
function escrow(uint256 _AuctionID)

// Setter
function setContracts(address aRegister, address aCoin, address aFee) onlyOwner
function setFrequency(uint256 _AuctionID, uint256 _frequency)

// Getter
function getFrequency(uint256 _AuctionID) returns uint256
function getTrafo() returns uint32
function getMCP(uint256 _AuctionID) returns uint64
function getAuctionID() returns uint256
```

Figure 5.3 – i7Chain Smart Contracts DoubleAuction.sol

Similar to the FAUCoin and Register contract, the owner variable stores the address of the contract deployer. The Reg, Coin, and Fee variables store the reference to instances of the Register, FAUCoin, and GridFee contract. The AuctionID variable is a counter that goes up in increments of 1 after every market clearing. It is used as a reference for every auction period. The Trafo variable holds the corresponding TrafoID for this double auction. The GridPriceBuy and GridPriceSell variables hold information regarding which prices the public grid buys and sells electricity for. The MinRate and MaxRate variables are initialized with the smallest possible value for MaxRate and a value close to the maximum value of a uint16 for MinRate. These variables are later used for identifying the market clearing price. The AuctionTime variable stores the auction period's duration in seconds. The constant booleans ModCongestionFee, ModAutarkFrequencyFee, ModSemiAutark-FrequencyFee, ModAutarkVoltageFee, and ModSemiAutarkVoltageFee determine which grid fee approaches apply to the market. The value "true" means that the corresponding fee applies. If the autark and semi-autark approach of a grid fee is set to true, only the autark grid fee will be applied. The mapping ParticipantsBids maps AuctionIDs to an array of the Details struct. It holds information pertaining to all bids for every auction period. The mapping ParticipantsUsage is similar in that it also maps AuctionIDs to an array of the Details struct. It holds information relative to all actual usages per auction period. The mappings BidderBool and UsageBool map the AuctionID to a mapping of addresses to booleans. They hold information about which of the addresses sent bids or which of the addresses submitted their actual usage information. The Frequencies mapping maps the AuctionID to the frequency level of the corresponding auction period. The MCPs mapping maps the AuctionID to the market clearing price for the corresponding auction period, and the Timer mapping maps the AuctionID to the unix timestamp of the last market clearing. The SRate and DRate arrays are empty arrays which will be used to store the supply and demand bids in an easily accessible manner.

The contract also contains six events. The NewBid event is emitted when a new bid is registered. The NewMcp event is emitted when a new market clearing price is determined. The NewUsage event is emitted when a new actual usage is registered. The EscrowEvent is emitted during the escrow function of this contract and relays information about the market's actual usage during the auction period. The Transfer event is emitted when money is transferred within or outside of the LEM. The TimerEvent is emitted when a new bid is processed, and returns whether a new auction period already started or not. The Details struct is used for bidding and the actual usage, and it holds information relative to price, amount, energy type, sender address, and a second amount variable, which is helpful for the escrow function. When the contract gets deployed, the constructor is called, and in the

constructor, the owner variable, TrafoID, first AuctionID, GridPriceBuy, GridPriceSell, and the first Timer are set.

In addition, the contract contains five modifiers that restrict the calling of functions. The onlyOwner modifier is similar to the onlyOwner modifier in the FAUCoin or Register contract. The onlyValidProsumer modifier checks if the message sender has the right energy type, is registered at the correct trafo, and has sufficient balances in its account in order to send a bid to the auction. 100 Euros were defined to be the threshold for sufficient funds, which is pretty high considering that an average transaction for a 15 minute auction period values less than 50 cents, but for testing purposes, the threshold was set higher. The onlyValidBidders modifier checks if the message sender has already sent a bid for the corresponding AuctionID, as a means to send an actual usage for this AuctionID. The onlyOneBidPerPeriod modifier checks if the message sender has already sent a bid for the corresponding AuctionID. If the message sender did already send a bid to the corresponding AuctionID, the transaction will be reversed. The onlyOneUsagePerBlock modifier functions the same way as the onlyOneBidPerPeriod modifier, but for the actual usage submission.

The checkLabel function determines whether an energy type is a producing or consuming energy type, subject to the passed-on integer value. If the passed-on value is between 0 and 9, it is a producer and 0 is returned. If the passed-on value is between 10 and 19, it is a consumer and 1 is returned. In case a higher value is passed-on to the function, 2 is returned.

The preBid function is called by the agents for submitting their bids. It is restricted by the onlyValidProsumers modifier and checks if a new auction period is reached by verifying if the current block's timestamp is larger than the sum of the saved timestamp of the last market clearing block and the auction period time. If applicable, the market becomes cleared and the second-to-last auction period becomes finalized by the escrow function. In addition, TimerEvent is emitted. If not applicable, only TimeEvent is emitted, but with a different string for the output parameter. In any case, the bid function is automatically called afterwards with the same parameters that are passed-on to the preBid function. Figure 5.4 shows an example of an agent attempting three bid submissions where only two of them are successful, one leading to a marketClear and escrow function call. This function is implemented in such a way that the first agent who sends a bid to a new auction period automatically calls the marketClear and escrow functions, and therefore has to pay for the consumed gas. This is reviewed again in Chapter 6.

The bid function takes amount, price, and energy type as input parameters. It is restricted by the onlyOneBidPerPeriod modifier, and internal, so that it can only be called by the contract itself (at the end of the preBid function). It takes the input parameters and stores them together with the address of the message sender in a Details struct. This struct becomes attached to the Details array that is mapped in the

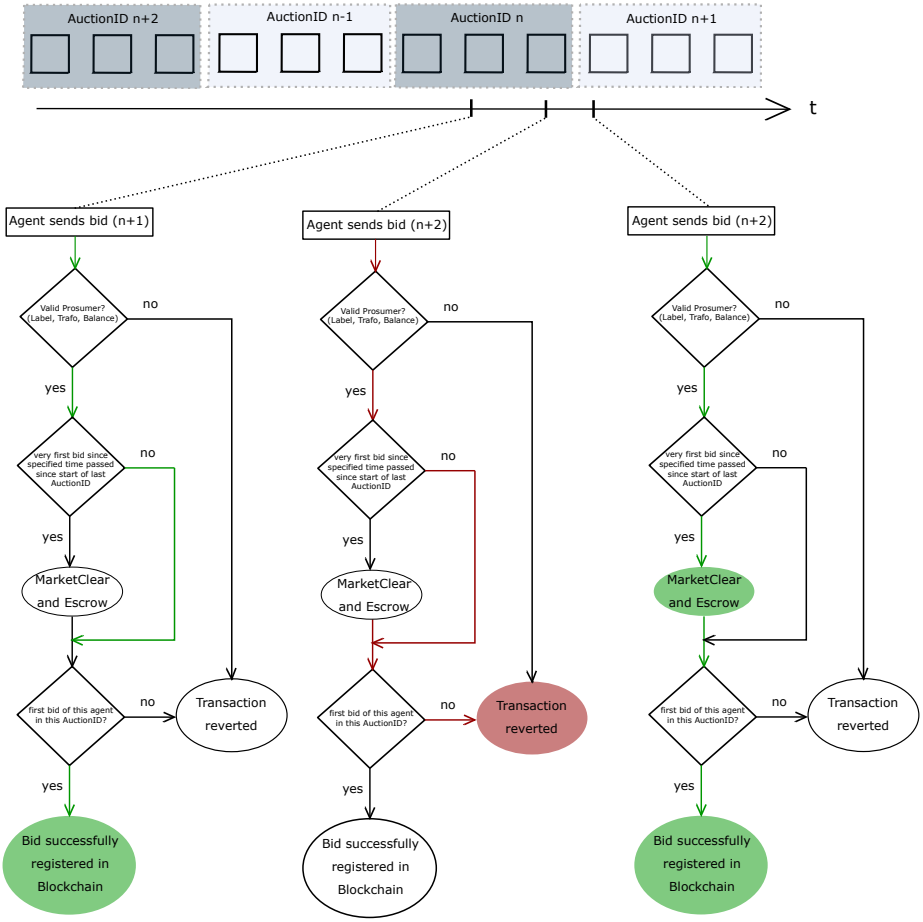


Figure 5.4 – Example of a bidding sequency

ParticipantsBids mapping. The mapping is based on the current AuctionID. Moreover, the address of the message sender is set to true in the BidderBool mapping at the current AuctionID, which later proves necessary for the onlyValidBidders modifier to clarify if the address has sent a bid in the specific AuctionID. Afterwards, the passed-on price is compared to the MaxRate and MinRate variable of the contract. If it is larger than the MaxRate value, the MaxRate variable is set to the passed-on price, and if it is smaller than the MinRate value, the MinRate variable is set to the passed-on price. The MinRate and MaxRate variables are used in the marketClear function. A for-loop iterates over all submitted bidding prices, and the MinRate and MaxRate variables help keep the loop as small as possible in order to conserve gas. The bid function continues by emitting the NewBid event and adding the passed-on amount to either the SRate or DRate array, depending on the energy Type. The value is added to the array using the passed-on price as index.

The marketClear function determines the market clearing price for the most recent auction period. It is only called from the preBid function. First, the Max variable is initialized with the MaxRate value and the y value is declared, which will be used for iterations in the for-loops. The first for-loop iterates over a range of MinRate to MaxRate. It cumulates the amount of producing bidders in the SRate array in ascending order, and the amount of consuming bidders in the DRate array in descending order. The SRate array and DRate array can be seen now as supply and demand curves. The point at which the two curves intersect represents the market clearing price (also called the equilibrium price) that this function is trying to determine. The second loop finds this intersection by iterating over the range of MinRate to MaxRate, while simultaneously checking each iteration to see if the SRate value at the current iteration index is greater than or equal to the DRate value at the current iteration index. If applicable, the following if-clause checks whether the DRate at the current iteration index is greater than the SRate value at the last iteration index, so as to make sure that a sufficient supply is available for the corresponding demand. If the if-clause condition is true, the current iteration index is set to be the market clearing price in the MCPs mapping based on the current AuctionID. If the condition is not true, the last iteration index is set to be the market clearing price. Regardless of the outcome, the NewMcp event is then emitted and the reset function is called before the marketClear function returns the market clearing price and terminates the function. If no market clearing price was found (SRate is lower than the DRate value at every point), an if-clause checks if the current AuctionID is greater than 1. In this case, the market clearing price is set to 13.0 cents. If it is not the first auction period and still no market clearing price has been found, the new market clearing price is set to the last market clearing price, plus 1.0 cent. This happens when only consuming (and no producing) bids were submitted for the last auction period. That is why the market automatically decides to raise the clearing price; a prominent energy deficit obviously exists. If no consuming bids and only producing bids are submitted, the MCP is automatically found at the start of the second for-loop, because the SRate is always greater than the DRate (within the MinRate/MaxRate range). In addition, the NewMcp event is emitted and the reset function is called before the market clearing price is returned.

The reset function sets all the variables that became filled during the last auction period back to their initial state and starts the new auction period. It iterates over the SRate and DRate arrays in order to set all values at every index to 0. It also adds 1 to the current AuctionID value and stores it again in the AuctionID variable. Furthermore, the Timer mapping for the new auction period is updated and the current blocks timestamp is stored in it. Ultimately, MinRate and MaxRate are set back to their initial values of 65000 and 0.

The actualUsage function is called by agents in order to submit their actual consumption or production amounts, along with the energy type, voltage level, and information relevant to which AuctionID it corresponds to (usually the AuctionID from two auction periods ago). The function is restricted by the onlyValidBidders and onlyOneUsagePerBlock modifiers. At the beginning, a Price variable and an Amount variable are declared as unsigned integers. A for-loop then iterates over the entire ParticipantsBids mapping in order to find the bid sent by the same address as that of the agent calling this function (the message sender). If the address is found, which is secured by the onlyValidBidders modifier, both the price and amount of the bid become stored in the variables that were declared at the beginning of this function, and the for-loop terminates. Afterwards, the passed-on amount and energy type, as well as the determined bidding price and amount, are stored together with the message sender's address in a Details struct and added to a Details array, which is mapped to the passed-on AuctionID in the ParticipantsUsage mapping. The message sender's address is also set to true in the UsageBool mapping at the corresponding AuctionID and the NewUsage event is emitted. Ultimately, the voltage information is passed on to the GridFee instance together with the message sender's circuit information and the AuctionID that was passed on to the actualUsage function. Everything is used in the GridFees registerVoltage function, which is explained later in this section.

The innerTransferMoney function is internal and only called by the escrow function. It is responsible for transferring the correct amounts of money for a successful interaction at the market, depending on the market clearing price and applicable grid fees. First, three variables are declared as integers. Then, the function checks which approaches are activated for this market. Either the autark approach, semi-autark approach, or simply no additional fee is applied. This happens for the frequency and voltage fees by checking if either of the autark or semi-autark constants is set to true. If both are set to true, only the autark approach applies. If neither is set to true, no additional fees are added. Depending on the chosen approach, the corresponding function is called in the GridFee instance and the necessary parameters are passed on. The functions called will determine, based on the passed-on parameters, the value of the FrequencyFee and VoltageFee variables. These values can also be negative seeing as though producers are punished by receiving less money and consumers are punished by paying more money. Furthermore, participants cannot only be punished, but also rewarded for their grid-friendly behavior. After all, the amount value, which is passed on to the innerTransferMoney function, is multiplied by the market clearing price, which is determined over the MCPs mapping and the passed-on AuctionID. The FrequencyFee and VoltageFee are added to this value and the result is stored in the reward variable. Afterwards, an if-clause checks whether the corresponding

participant is a consumer or producer by calling the checkLabel function with the passed-on energy type as a parameter.

If the participant is a producer, the changeCoinBalance function is called in the FAUCoin instance with the passed-on address and reward variable as parameters. If the participant is a consumer, the same process occurs, but with a negated reward variable. The Transfer event is also emitted. At this point, the participant has a new account balance and has either paid for grid fees or received money for grid-friendly measures. Now, the frequency and voltage account must be updated. Therefore, the changeCoinBalance function of the FAUCoin instance is first called with the addresses from the frequency account (which is determined through the getTrafoFrequencyAddr function of the Register instance) and the FrequencyFee variable as a parameter. Afterwards, the same is done for the voltage account, but the parameters for the changeCoinBalance function are obtained by the getTrafoVoltageAddr function and the VoltageFee variable.

The outerTransferMoney function is internal and only called by the escrow function. It is responsible for transferring the correct amounts of money for an unsuccessful interaction at the market. It is similar to the innerTransferMoney function, differing in the fact that it will add a congestion fee in addition to the frequency and voltage fee, and change the balance of the congestion account if the ModCongestionFee is set to true and a congestion is applied. Furthermore, it changes the balance of the public grid account because this is where it's either procuring the energy from or selling the energy to, based on the GridPriceSell or GridPriceBuy.

The escrow function is the most extensive function in the DoubleAuction smart contract. It is called from the preBid function and takes an unsigned integer, which represents an AuctionID, as a parameter. It calculates the total amount that has been produced and consumed, sets the trafo load based on the difference between these amounts, and then starts to assign the money transfers. In the beginning, an unsigned integer variable for the following for-loops is declared, as well as three more unsigned integer variables called AmountProduced, AmountConsumed, and Flag. The first for-loop iterates over the entire Details array that is mapped to the passed-on AuctionID in the ParticipantsUsage mapping. At every index, it checks if the current Details struct holds information on a producer or consumer. Based on this information, it adds the amount that is stored in the Details struct to either the AmountProduced or AmountConsumed variable. After the for-loop, the AmountProduced and AmountConsumed variables are compared and the Flag variable is set. If AmountProduced is larger than AmountConsumed, the Flag is set to 0. If AmountConsumed is larger than AmountProduced, the Flag is set to 1. If both values are equal, the Flag is set to 2. Afterwards, the EscrowEvent is emitted with the flag information, the AmountProduced and AmountConsumed values, the number of participants (length of Details array in ParticipantsUsage mapping at

passed-on AuctionID), and the AuctionID. The function then continues on to set the trafo load. Based on the Flag value, the difference between the AmountProduced and AmountConsumed variables becomes calculated in a way where the result is always positive. This result is passed on to the setCurrentTrafoLoad function in the GridFee instance together with the Flag value. This is important in enabling the GridFee instance to later be able to decipher whether too much energy has been produced or consumed in the LEM. In addition, the setAmounts function of the GridFee instance becomes called with AmountProduced and AmountConsumed as parameters.

To assign the money transfers, two more unsigned integer variables are declared. The AmountDiff variable, which is initialized with a value of 0, and the FlagDiff variable, which is initialized with a value of 2. The function iterates again over the entire Details array that is mapped to the passed-on AuctionID in the ParticipantsUsage mapping. For each iteration, an if-clause checks if the actual used amount (amount variable of Details struct) is greater or less than the bidding amount (amount2 variable of the Details struct). Based on this if-condition, the AmountDiff value is calculated by subtracting the smaller value from the bigger value and the FlagDiff variable is set to either 0, for a larger actual used amount, or to 1, for a larger bidding amount. In the case of equal amounts, the FlagDiff value stays at its initialized value of 2 and the AmountDiff stays at 0. Afterwards, if-clauses check whether the current Details struct holds information on a producer or consumer, which is very important because producers receive money and consumers must pay money. The following procedure is first explained for producers.

The next thing checked is whether the bidding price of the producer is lower than, higher than, or equal to the market clearing price. If it is higher, the producer has to trade outside of the LEM, which means the outerTransferMoney function is called with the necessary parameters from the Details struct. If the price is lower or equal to the market clearing price, the producer can trade its bidding amount within the LEM and has to procure or sell the difference between the bidding and actual usage amount to outside of the LEM. This is where the FlagDiff variable comes in handy because it already holds information on whether the producer has to procure additional energy or sell energy to the public grid.

In any case, the innerTransferMoney function is called with the bidding amount as a passed-on parameter because the producer committed to delivering this amount by sending its bid to the auction. Moreover, because the market clearing price was calculated based on the bidding amount, any deviation from these amounts would lead to an unbalanced auction. The outerTransferMoney function is then called with the AmountDiff variable as a passed-on parameter, and in case the producer has to sell additional energy, the passed-on energy type is changed to 18, which is a consumer label and not used for any specific energy type yet (as seen in Table 5.2).

In the rare case that the bidding amount equals the actual used amount, no trade outside the LEM is necessary; only the innerTransferMoney function is called.

This procedure works similarly for consumers. However, contrary to the producers, consumers are not successful at the auction and have to acquire everything from outside the LEM when they bid less than the market clearing price. In case they do bid more, the differences between the bidding and actual amount have to be traded with the public grid again. At the end of this function, the emitTrafoBalances function from the Register instance is called.

The setContracts function is restricted by the onlyOwner modifier and is a simple setter function that sets the reference to instances of Register, FAUCoin, and GridFee, based on the passed-on addresses of these contracts.

The setFrequency function is called from outside for every auction period in order to set the current frequency level, which is vital in calculating the frequency grid fees. It is assumed that the trafo measures the frequency during each auction period and then calls this function. The value then gets stored in the Frequencies mapping based on the passed-on AuctionID.

The getFrequency, getTrafo, getMCP, and getAuctionID functions are simple getter functions that return the corresponding value. The getFrequency and getMCP functions need an AuctionID as a parameter, while the other two getters are not dependent on a specific auction period.

5.3.4 GridFee

The GridFee smart contract holds all the information necessary for calculating the different grid fees. It mainly receives information on the grid from the DoubleAuction contract. Aside from calculating the fees, it also stores boundaries for the different fee-levels and corresponding rates. Figure 5.5 shows the class diagram of the GridFee smart contract.

The owner variable stores the address of the contract owner and the Trafo variable stores the corresponding TrafoID. Both are set in the constructor. The Reg variable stores an instance of the Register smart contract. The CurrentTrafoLoad variable is used to store the current load on the trafo, which is the difference between the produced and consumed amount in the LEM. The TrafoFlag variable holds information on whether the trafo has to obtain energy from or sell energy to the public grid. The AmountProduced and AmountConsumed variables are declared in order to later hold information stating the total amounts that were produced and consumed in the LEM. The two CongestionMaxLevel constants store the thresholds for the congestion fee-levels. Within 50%, CongestionLevelRateOne is applicable, within 80%, CongestionLevelRateTwo is applicable, and beyond that, CongestionLevelRateThree is applicable. The CongestionLevelRate values store information relative to how

GridFee.sol

```
address public owner

Register Reg

uint32 Trafo
uint32 CurrentTrafoLoad
uint8 TrafoFlag
uint32 AmountProduced
uint32 AmountConsumed

uint8 constant CongestionMaxLevelOne = 50
uint8 constant CongestionMaxLevelTwo = 80
uint8 constant CongestionLevelRateOne = 0
uint8 constant CongestionLevelRateTwo = 10
uint8 constant CongestionLevelRateThree = 25

uint8 constant FrequDiffMaxLevelOne = 80
uint8 constant FrequDiffMaxLevelTwo = 140
uint8 constant FrequLevelRateOne = 0
uint8 constant FrequLevelRateTwo = 10
uint8 constant FrequLevelRateThree = 25

uint32 constant VoltDiffMaxLevelOne = 500
uint32 constant VoltDiffMaxLevelTwo = 1000
uint8 constant VoltLevelRateOne = 5
uint8 constant VoltLevelRateTwo = 10
uint8 constant VoltLevelRateThree = 15

mapping (address =>bool) ContractAddress
mapping (uint256 => mapping(uint8 => VoltageReg)) VoltageDir

event ShowCongestionFee(uint32 _currentTrafoLoad, uint32 _MaxTrafoLoad, int _congestionFee, int usage)
event ShowFrequencyFee(uint256 frequDiff, int fee, uint32 amount, uint8 reallabel, uint256 Frequency)
event ShowVoltageFee(int fee,uint32 amount, uint8 reallabel, uint64 voltdiff, uint32 voltlevel, uint256 AuctionID, uint8 circuit,
... bool direction)
event EvenVoltReg(uint32 _voltage, uint256 _AuctionID, address sender, uint64 _diff, bool direction)
```

```
struct VoltageReg {uint64 diff; bool direction; }

modifier onlyOwner()
modifier onlyValidContract()

constructor()

function congestionFee(uint8 _reallabel) returns int32
function autarkFrequencyFee(uint32 _amount, uint8 _reallabel, uint256 _Frequency) returns int
function semiAutarkFrequencyFee(uint32 _amount, uint8 _reallabel, uint256 _Frequency) returns int
function autarkVoltageFee(uint32 _amount, uint8 _reallabel, uint256 _AuctionID, uint8 _circuit) returns int
function semiAutarkVoltageFee(uint32 _amount, uint8 _reallabel, uint256 _AuctionID, uint8 _circuit) returns int
function frequencyLevel(uint256 _frequDiff) returns uint8
function voltageLevel(uint64 _voltDiff) returns uint8
function registerVoltage(uint8 _circuit, uint32 _voltage, uint256 _AuctionID) onlyValidContract

// Setter
function setContractAddress(address _contract, bool _tf) onlyOwner
function setRegister(address addr) onlyOwner
function setCurrentTrafoLoad(uint32 _currentLoad, uint8 flag) onlyValidContract
function setAmounts(uint32 _amountProduced, uint32 _amountConsumed) onlyValidContract
```

Figure 5.5 – i7Chain Smart Contracts GridFee.sol

many cents will be added per kwh. The value 10 equals 1.0 cent. The frequency and voltage constants function similarly. The frequency deviation thresholds are 0.08 Hz (FrequDiffMaxLevelOne = 80) and 0.14 Hz (FrequDiffMaxLevelTwo = 140). The voltage deviation thresholds are 5 V (VoltDiffMaxLevelOne = 500) and 10 V (VoltDiffMaxLevelTwo = 1000). The values that are set here are described in more detail in Section 6.1.1. The ContractAddress mapping maps addresses to booleans and the VoltageDir mapping maps AuctionIDs to a mapping of circuits to

the VoltageReg struct. The VoltageReg struct contains an unsigned integer for a deviation of the current voltage value from 230V, as well as a boolean that is true for high voltage and false for low voltage.

The contract also contains four events. The ShowCongestionFee, ShowFrequencyFee, and ShowVoltageFee event all return necessary information pertaining to the specific fee. The EvenVoltReg event is emitted when a new voltage is registered, which has a higher deviation from 230V than the old one.

The onlyOwner modifier restricts a function to only be callable when the message sender has the same address as the owner variable, and the onlyValidContract modifier restricts a function to only be callable when the message sender's address is set to true in the ContractAddress mapping.

The congestionFee function takes a label as input parameter and returns the congestion fee rate, which is later multiplied by the amount in the outerTransferMoney function of the DoubleAuction contract. The function first calculates the usage of the trafo by dividing the current load by the maximum load of the trafo. Since solidity cannot handle floats and doubles yet, the current load is first multiplied by 100, allowing the result of an integer. Afterwards, the usage is compared to the different thresholds and, based on this comparison, the CongestionFee variable is filled with the corresponding rate. The ShowCongestionFee event is emitted and, ultimately, the function checks whether the congestions fee must be negated or not. If the passed-on label is 0, which represents a producer, and if the TrafoFlag is 1, which means that the LEM has a deficit, then the CongestionFee must be negated. This negated amount eventually leads to a higher reward for the producer in the outerTransferMoney function due to the energy produced during times of an energy deficit. If the passed-on label is 1, which represents a consumer, and if the LEM produces too much energy, the CongestionFee variable is negated too because a consumer is then rewarded for consuming and actively doing something against the surplus of the LEM. In the end, the CongestionFee variable is returned.

The autarkFrequencyFee function takes an amount, a label, and a frequency as input parameters and returns the total frequency fee in the end, which is later added to the reward variable in the inner and outerTransferMoney function of the DoubleAuction contract. It starts with checking whether the passed-on frequency value is higher or lower than 50 Hz. If it is higher, producers of energy should be punished, and if it is lower, consumers of energy should be punished. If it is higher, the difference between the passed-on frequency and 50 Hz is calculated. Then, an if-clause checks if the passed-on label is a producer or consumer label. If it is a producer label, the frequency fee is calculated by multiplying the passed-on amount by the returned value of the frequencyLevel function, which takes the previously calculated difference as an input and outputs the fee rate. The Fee variable also becomes negated. If it is a consumer label, the procedure is a little more complicated

because this is an autark approach function. That means that only non-grid-friendly participants pay the fixed-fee-level rates and the participants that should be rewarded receive a partial amount of the money that was collected from the non-grid-friendly participants. Subsequently, a ratio must first be calculated. The passed-on amount, which is multiplied by 100, is divided by the total amount that was consumed. This ratio is then multiplied by the amount that was produced and the corresponding frequency level rate before it is divided by 1000. The resulting Fee variable must then be negated again. If the passed-on frequency is lower than 50 Hz, the whole procedure is flipped and the fee variables must not be negated. At the end of the function, the ShowFrequencyFee event is emitted and the Fee variable is returned.

The semiAutarkFrequencyFee function is a bit more simple than its autark counterpart. It has the same input parameters and functionality as the autark approach, but differs through the fact that the participants who act grid-friendly do not receive a partial amount of the fees. These participants also receive the traded amount multiplied by the frequency fee rate as a reward.

The autarkVoltageFee function is very similar to the autarkFrequencyFee function. However, instead of checking if the frequency is higher or lower than 50 Hz, this function checks if the VoltageReg struct that is mapped to the passed-on AuctionID and passed-on circuit has either a true or false direction boolean. If it is true, producers become punished, and if it is false, consumers become punished. Rather than the frequencyLevel function, the voltageLevel function is called. At the end, the ShowVoltageFee event is emitted and the Fee value is returned.

The semiAutarkVoltageFee function is a combination of the semiAutarkFrequencyFee function and the autarkVoltageFee function.

The frequencyLevel function, which is only called by the autarkFrequencyFee or semiAutarkFrequencyFee function, takes the deviation from the standard frequency level as an input parameter, and based on this parameter, it returns the fee-level rate. This happens through comparing the passed-on deviation and the frequency boundaries, which are defined as constants at the beginning of the contract.

The voltageLevel function, which is only called by the autarkVoltageFee or semiAutarkVoltageFee function, works the same way as the frequencyLevel function, but for voltage.

The registerVoltage function is necessary for identifying the highest voltage deviation in a circuit. It is called from the actualUsage function in the DoubleAuction contract and takes the circuit, voltage level, and AuctionID as input parameters. It checks if the passed-on voltage is higher or lower than the standard voltage level and then calculates the deviation from the standard voltage level. A boolean value is also set to true if the passed-on voltage is higher than the standard voltage level, or set to false, if it is lower. In the end, the calculated deviation is compared to the current highest registered voltage deviation. If the calculated deviation is higher, it becomes

stored together with the boolean value in a VoltageReg struct and attached to the VoltageDir mapping based on the passed-on AuctionID and circuit. Furthermore, the EventVoltReg event is emitted.

The contract ends with a number of setters. The setContractAddress, setRegister, setCurrentTrafoLoad, and setAmount setter. The last two setters are restricted by the onlyValidContract modifier, while the setContractAddress function is restricted by the onlyOwner modifier.

5.3.5 Market Initialization

The entire market is initialized by the initializeMarket python script. This script compiles the solidity code (using solc), connects to the blockchain via web3.py, and deploys the four smart contracts. Beyond that, it also calls the functions that set the other necessary contract instances in each smart contract and adds a trafo to the market. In the end, the smart contract addresses are stored in .txt files and the compiled solidity code is stored in a .json file.

5.4 Agents

Although agents are not the focus of this thesis, they prove necessary for successful simulations. In order to receive a plausible, meaningful outcome from the simulations, the agents were implemented to act as realistic as possible. All of the agent code is implemented in python 3.

Figure 5.6 shows the class diagram of the implemented classAgent class and the classReturnValue class, which is used by the agent class. Its function is to store what is identified as the most important information, storing it during each auction period, and later exporting it in a .csv file.

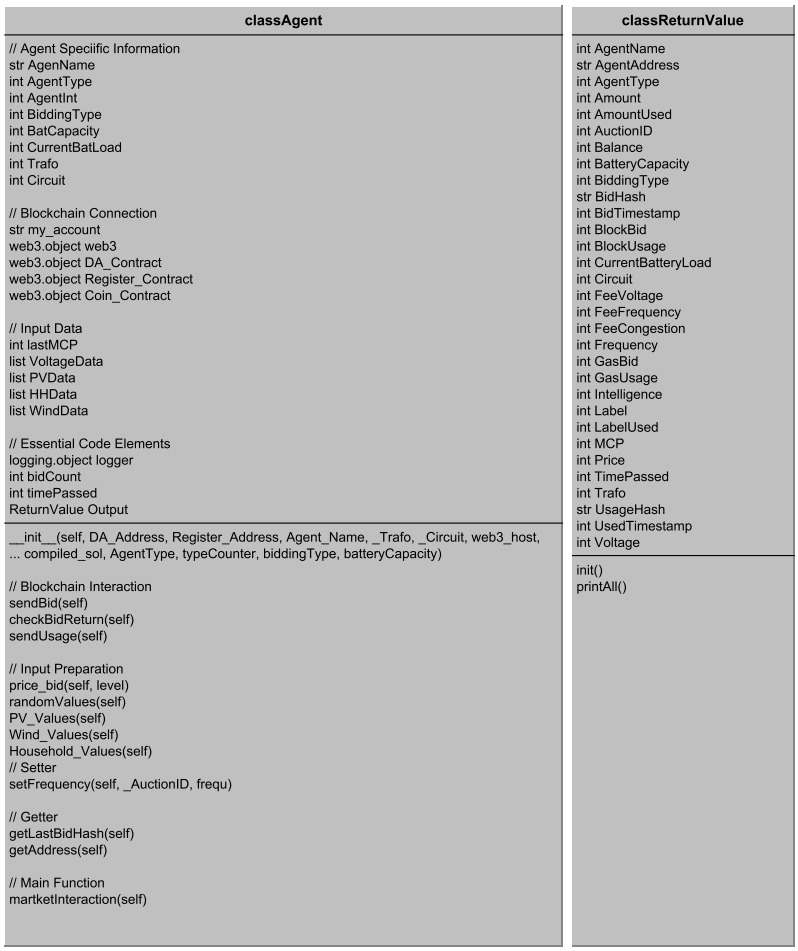


Figure 5.6 – Agent Class Diagram and ReturnValue Class Diagram

5.4.1 Agent Types

Different types of agents are implemented (household, photovoltaic, wind, random), all of which use different input values and have different labels. The input values of the random agents are, as the name implies, randomly generated from a range between 100 and 1500 Wh, with a label that is also randomly generated.The input values of the wind agents are based on Germany's total wind production and have been downscaled in order to fit our market. The data is offered by the Bundesnetzagentur [69]. The input values of the photovoltaic agents are provided by computer science 7. The input values of the households are based on the electrical load profiles of residential buildings in Germany [70]. For this thesis, the resolution of the data is aggregated to 15 minutes and the time period covered is over a duration of seven days. In an effort to create more realistic simulations, some household data was

tweaked, meaning that photovoltaic production amounts were subtracted from the household load in order to attain the data of households with photovoltaic. All input data is stored in separate .csv files, one file for each agent. Household agents can also have batteries where they can store energy. This allows the household agent to exist as the only agent able to offer flexibilities to the market. Wind and photovoltaic agents have to sell their electricity within the same auction period it was produced, however, household agents have the ability to choose between selling and buying energy at the market, or utilizing their batteries. For this reason, the most complicated implementation is that of the household agent.

The wind, photovoltaic, and random agents simply send bids to the market based on their input values. These bids are modified a bit in order to be more realistic because bids in real-life are merely predictions for the next auction period. The bidding price for these agents is based on their intelligence and bidding type. ZI agents don't have bidding Types, they simply send randomly generated bids from a range of 8.0 to 26.0 cents. Intelligent bidding agents bid based on gathered data and do have bidding Types. They can either send conservative, less conservative, or slightly more progressive bids. For producers, conservative bids are significantly lower than the last MCP, but for consumers, they are significantly higher than the last MCP. Less conservative bids are similar, except that the deviation from the MCP is smaller. Progressive bids are a little over the MCP for producers and a little under the MCP for consumers. Progressive bidders have the hope to find better deals, but on average, they are less often successful and must trade more frequently with the public grid.

Intelligent Household agents are more complicated in the sense that at every auction period, they must decide whether they should buy at the market, sell at the market, or use their batteries. They can also partially buy or sell the consumption/production at the market, using their batteries for the rest. Their decision is primarily contingent on their batteries State of Charge (SOC), but also based on the last MCP and the current voltage level in the grid (frequency and congestion is not implemented). If the SOC is high, so rises the chance that household agents sell more electricity at the market, whereas if it is low, the chance they will buy more electricity at the market rises. If the voltage is high, agents are also more likely to buy more electricity at the market because they will be rewarded by the grid. If it is low, they are more likely to sell more electricity. If they are not successful at the double auction, they will always decide to use their battery load, if possible, instead of buying from the public grid. Buying at the public grid can lead to additional congestion fees and is always possible, but only for comparatively high and fixed grid prices. Therefore, agents try to avoid interacting with the public grid. Their bidding price also varies depending on the aforementioned factors. More information on the

decision making of the intelligent agents can be found in the corresponding python script in Chapter B.

5.4.2 Blockchain Interaction

Interaction with the blockchain is possible through the web3.py library and mainly happens in the functions sendBid, checkBidReturn, sendUsage, and MarketInteraction.

The sendBid function is always called for the current auction period. It tries to call web3's estimateGas function for the preBid function of the DoubleAuction contract until it receives a result. The estimateGas function estimates the amount of gas a potential transaction on the blockchain would consume. Provided that the market is still in an old auction period (this agent already sent a bid for this period), the estimateGas function will fail because of the bid function's onlyOneBidPerPeriod modifier, and the agent waits between 1 and 3 seconds before making another attempt. If the estimateGas function is successful, the actual preBid function is transacted (together with the amount, price, and label as parameters) and the transaction hash is stored. In addition, the getCoinBalance function of the FAUCoin contract becomes called in order to receive the agent's balance. All of this information is stored in the corresponding ReturnValue instance.

The checkBidReturn function is always called for the last auction period. It uses the hash that was stored during the sendBid function and requests the transaction receipt for this hash from the blockchain. If the corresponding block has not been mined yet, the agent waits two seconds and tries to request the transaction receipt again. When the agent receives the transaction receipt, he processes it, looking for the NewBid event. While processing the receipt, the bid hash is looked up (in hexadecimal). If found, the AuctionID and MCP data are read and stored in the corresponding ReturnValue object.

The sendUsage function is always called two auction periods after the corresponding bid was sent. It is similar to the sendBid function, but instead of transacting the preBid function, the actualUsage function of the DoubleAuction contract is first estimated and then transacted.

The marketInteraction function organizes the function calls of the three other previously mentioned functions, while also adding some additional blockchain interaction (e.g. reading the transaction receipt of the sendUsage transaction). At the end of this function, a ReturnValue instance is returned. This ReturnValue instance contains the agent's bid from four auction periods ago, all the corresponding transaction receipts and usage data, as well as the balance and time that has passed in between the sent bid and the following bid.

5.4.3 Agent Initialization

An agent is initialized through the startAgent python script. The user first chooses the agent's type, input data, and intelligence. Then, the script collects any necessary blockchain information (such as web3 host, contract addresses, and compiled solidity code) from the files that were stored during the initializeMarket script. With this information, it initializes a new object from the Agent class and starts an endless while-loop. In this loop, it calls the MarketInteraction function of the Agent class for every auction period and stores the information from the returned ReturnValue objects in a .csv file.

5.5 Additional Scripts

In addition to the agent-specific scripts, more python scripts are implemented in order to gather additional data or automatize processes.

5.5.1 Market Listener

The MarketListener script connects to the blockchain via web3 and initializes the DoubleAuction and Register contracts as web3 objects. Afterwards, it iterates over the blockchain based on input values that mark the starting block and the ending block. Each block is checked for whether or not it contains transactions. If a block does contain transactions, the transaction receipt is requested and processed for the NewMcp, EscrovEvent, and TrafoBalances events. Out these events, it gathers the following information:

- AuctionID - The number of the current auction period
- MCP - The market clearing price
- minRate - The smallest bid or ask in the auction period
- maxRate - The highest bid or ask in the auction period
- MCPTime - The timestamp of the block that called the MarketClear function
- AmountProduced - The total amount produced for an auction period
- AmoundConsumed - The total amount consumed for an auction period
- NumberOfBids - The number of bids received
- BalanceCongestion - The balance of the congestion account
- BalanceFrequency - The balance of the frequency account

- BalanceVoltage - The balance of the voltage account

This information is stored in a .csv file through the function fillCSV. Each row represents information pertaining to one AuctionID.

5.5.2 Chain Listener

Similar to the MarketListener script, the ChainListener script also iterates over a predefined range of blocks. It outputs information about the blockchain itself. The following information is gathered:

- Blocknumber - The number of the current block
- TransactionCount - The number of transactions in the current block
- GasUsed - The amount of gas that was spent in the block
- GasLimit - The gas limit in the block
- Difficulty - The difficulty in the block
- Miner - The address of the block's miner
- Size - The size of the block
- TimeStamp - The time at which the block was mined

This information is stored in a .csv file through the function fillCSV. Each row represents information about one block.

5.5.3 CSV Grabber

Every agent stores its own .csv file. For 20 agents, 20 .csv files must be collected and summarized. This is what the grabCSV script does; it collects all .csv files in a folder and adds each row (apart from the header) from each .csv file to a new .csv file. The header is subject to the hard-coded columnNames list.

Chapter 6

Analysis

In this chapter, the implemented market and agents are used for market simulations, and the results of these simulations are presented and discussed. Furthermore, the blockchain's performance is reviewed and discussed.

6.1 Market Simulation

This section defines different market scenarios and their simulations. A market overview is given and, afterwards, the different scenarios and their results are shown.

6.1.1 Market Overview

Each simulation in this section uses the i7Chain with a 5-second block time and a 30-second auction period length as their basis. The input data covers 7 days (from Monday to Sunday) and the resolution is aggregated to 15 minutes. This means that simulating 7 days with 15 minutes of granularity only takes about 5.5 hours on the i7Chain, which clears every 30 seconds. Of course the use of input data with 30 seconds of granularity is possible as well (which would equal a real-time simulation), but no additional market simulation is conducted in this thesis with 30-second input data.

Due to the use of blockchain as an information layer, it is possible that nodes do not synchronize fast enough with the rest of the network, which can cause agents to miss sending a actual usage information for an auction period, because they receive the necessary transaction receipt too late. This needs to be mentioned because all of the following simulations use the same input data. However, because of the previously mentioned challenge regarding synchronisation, the different market inputs in terms of the actual usage may still slightly vary. The bid information is still

send correctly, which means that MCP and other information that relies on the bid information is correct and comparable.

For the scenarios, input data for 17 household agents (all with battery and 14 with photovoltaic), as well as 3 wind agents, was prepared. The total supply data can be seen in Figure 6.1 and the total demand data can be seen in Figure 6.2. Figure 6.3 shows the difference between supply and demand.

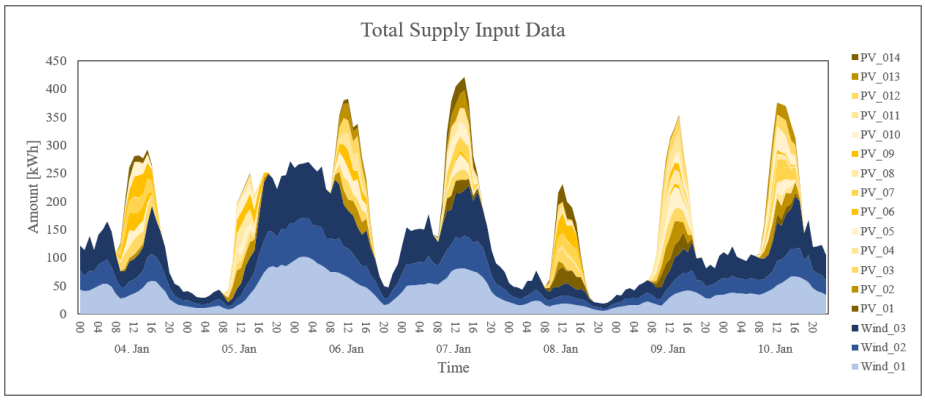


Figure 6.1 – Total Supply Input Data

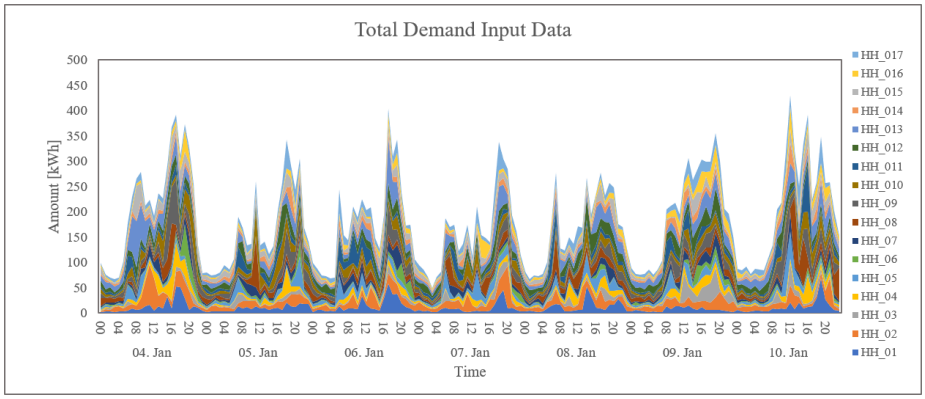


Figure 6.2 – Total Demand Input Data

In the following sections, the same solidity (market) and python (agent) codes are used, but with different parameters. The DoubleAuction contract, for example, holds information about which fees are applied and which approach is used. This information is altered according to different simulation purposes. The smart contracts are newly deployed for every new simulation case.

The grid fees in the GridFee contract are set as illustrated in Figure 6.4.

The thresholds of the congestion fee are based on the maximum trafo load (which is assumed to also reflect the maximum load of the connected transmission lines). The thresholds for the frequency fee depend on the frequency's absolute deviation

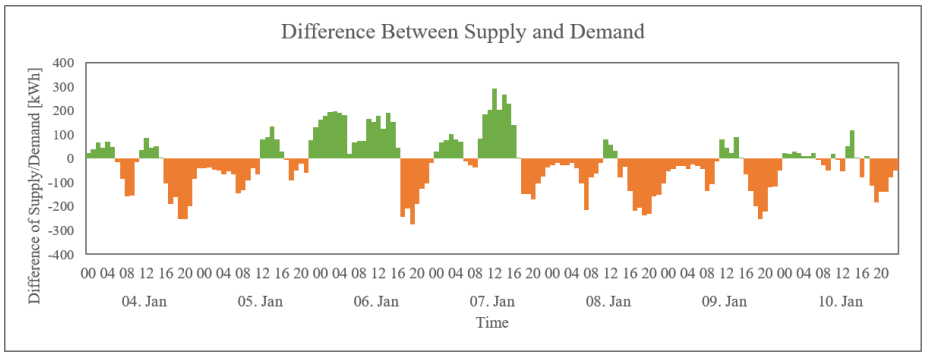


Figure 6.3 – Difference Between Supply and Demand

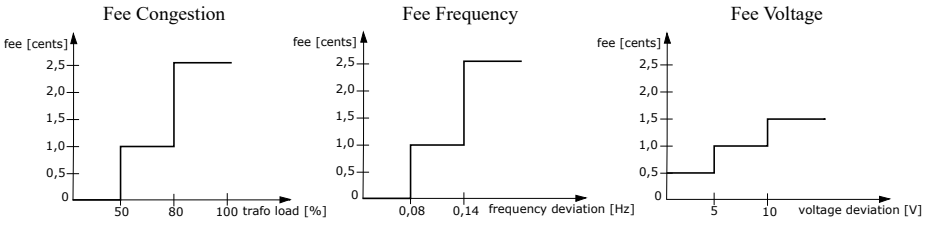


Figure 6.4 – Grid Fee Thresholds and Fee Rates

from the standard frequency of 50 Hz. They are defined based on the activation of control energy, as explained in Section 2.1.2. The thresholds for the voltage fee correspond to the voltage's absolute deviation from the standard voltage of 230V, and are defined by the fact that most electronic devices are only able to cope with a voltage deviation of 5 - 10%.

The fee rates are mainly based on the current tariffs paid for using the public grid. According to [71], the current charges for using the public grid are about 7.5 cents, which make up approximately 25% of the entire costs that are charged per kWh in Germany. Because the implementation of the LEM in this thesis is lean and no intermediaries have to be paid, the maximum amount of grid fees an agent has to pay is set to 6.5 cents (2.5 + 2.5 + 1.5). The parametrization of the grid fees is roughly based on the aforementioned assumptions but is not the focus of this thesis. The focus is rather on proving that the technical implementation of grid-influencing measures is possible.

Taxes, such as value added tax and electricity tax, are about 23% of the current charges per kWh [71]. For the implementation of a prototype with focus on grid-friendliness and blockchain technologies, these figures are not important, because they do not affect the market. Therefore, they are neglected in the simulation. However, they easily can be implemented as smart contracts. The share of the "EEG-Umlage" per kWh, is also about 23%. It is also neglected in the implemented market, because the "EEG-Umlage" pays for the difference between feed-in tariffs

and the actual amount electricity is sold for at the exchanges. This does not effect the implemented LEM, since no feed-in tariffs are offered. However, the interaction with the public grid contains all the common expenses, as mentioned in [71].

6.1.2 Comparison of No Fee Scenario to Semi-Autark Approach

Three main scenarios, were defined and simulated for this part of the thesis:

- Scenario 1: No fees applied.
- Scenario 2: Only semi-autark voltage fee applied.
- Scenario 3: Congestion and semi-autark voltage fee applied.

Theses three scenarios were defined in order to see the differences between the markets reactions; when no fees are applied in comparison to when voltage fee is applied. The scenario that adds congestion is not that important, because the agents cannot react yet to different congestion levels. It can be seen as an additional reference for the semi-autark voltage fee scenario. No frequency fee scenario is compared here, because the functionality is equal to the functionality of the voltage fee, hence, the implementation of the necessary agent code is not part of this thesis.

Scenario 1 is discussed first. It primarily represents the price creation at the market based on the current supply and demand input, but also based on the flexibility (the battery load) of household agents. Figure 6.5 illustrates the MCP for every auction period. Some interesting data points are highlighted and will be briefly discussed.

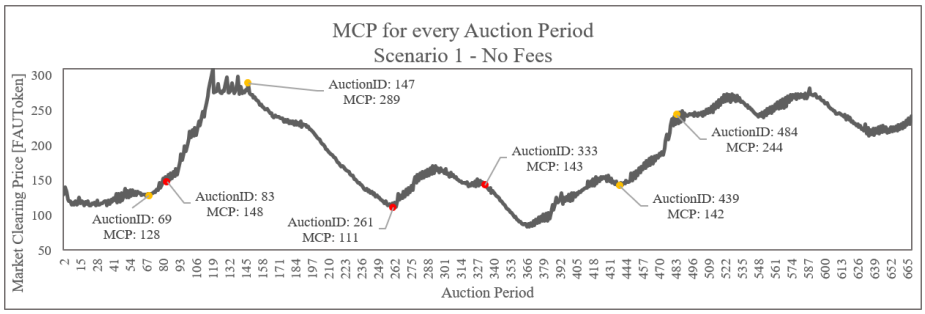


Figure 6.5 – MCP for every Auction Period - Scenario 1

Figure 6.5 shows that the MCP ranges from 8.0 cents to 30.0 cents. It is mainly influenced by the difference of the current supply and demand, which is shown in Figure 6.3. Figure 6.6 shows the difference between supply and demand of the submitted bidding amounts. For the wind agents, the bids almost match the input data, they are only altered a little bit in order to make it more realistic that bids

are predictions. For the household agents, the bidding amount depends on their intelligence, on demand/supply, as well as on their batteries current SOC.

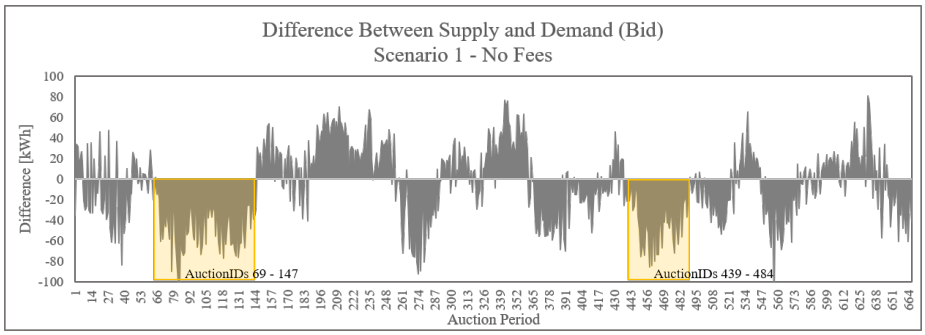


Figure 6.6 – Difference Between Supply and Demand (Bid) - Scenario 1

The first two yellow points in Figure 6.5 represent the starting and ending point of a steep incline of the MCP. It resulted from a high demand between AuctionID 69 and 147, which is highlighted in Figure 6.6. The MCP stopped rising at about 28.0 cents and hit a limit, because at that point it is cheaper for consumers to trade with the public grid. The third and fourth yellow point in Figure 6.5 represent the starting and ending point of a similar incline, which is not as steep as the first one mentioned. It is also highlighted in Figure 6.6.

Figure 6.7 illustrates the current batteries SOC for every auction period. The dashed lines indicate the SOC band.

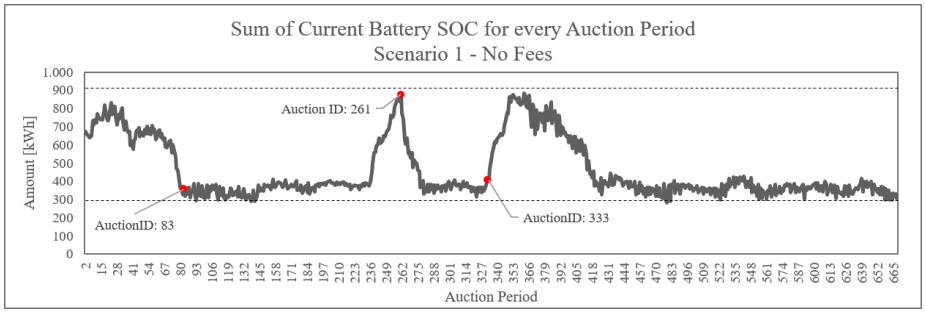


Figure 6.7 – Current Battery SOC for every Auction Period- Scenario 1

Figure 6.7 shows that at AuctionID 83, the sum of all current battery SOCs in the market was low compared to earlier AuctionIDs, which led to a steep incline of the MCP once the demand became higher than the supply. Before AuctionID 83, the MCP stayed nearly constant, even though the demand already exceeded the supply from AuctionID 69 onwards. It stayed constant because the household agents used their batteries to compensate for the high demand, which can be seen in the steep decline of the current battery SOC before AuctionID 83, in Figure 6.7.

Figure 6.7 also shows that the household agents refill their batteries repeatedly, especially during times of low MCPs and high supply, as seen at AuctionID 261. The MCP is at 111, which is comparatively low, and the current battery load is at its maximum. Shortly afterward, the MCP rises and the current battery load decreases, because demand exceeds supply. At AuctionID 333, the MCP starts falling again and the current battery load rises because supply exceeds demand.

The other two scenarios are not discussed as detailed as Scenario 1, but they are compared to it. Figure 6.8 shows the voltage deviation from 230V for the scenarios with voltage fees.

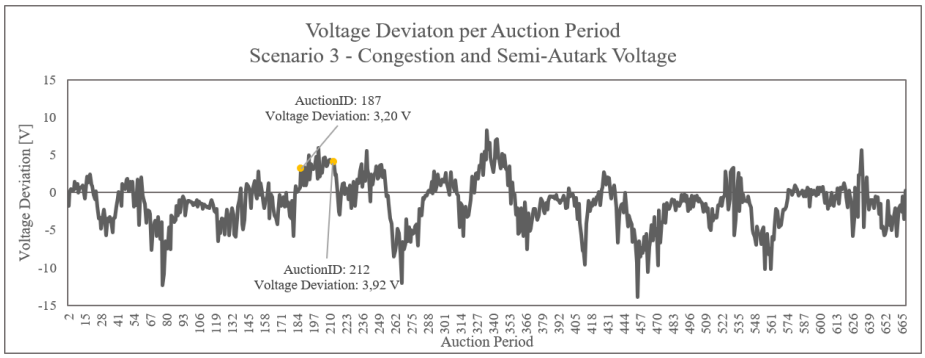


Figure 6.8 – Voltage Deviaton per Auction Period - Scenario 3

Figure 6.9 shows the MCPs of all three scenarios. It is important to remember that in case of the scenarios with fees, the MCP is not the final trading price. For these scenarios, a fee will be added on top of the MCP.

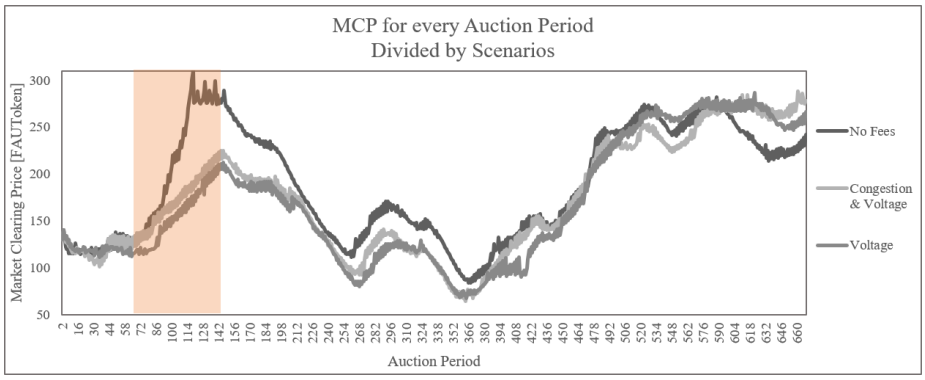


Figure 6.9 – MCP for every Auction Period - Divided by Scenarios

The curves in Figure 6.9 are all very similar, but between AuctionID 70 and 145 they can clearly be distinguished from one another. The MCP of Scenario 1, with no fees, rises much faster than the MCPs of the other scenarios. This is caused by the submitted demand of the households agents in Scenario 1. It is higher than

the submitted demand of the household agents in the other scenarios. Figure 6.10 visualizes the difference between demand and supply for Scenario 1 and Scenario 3. It can be seen, that the results vary, especially for the highlighted period.

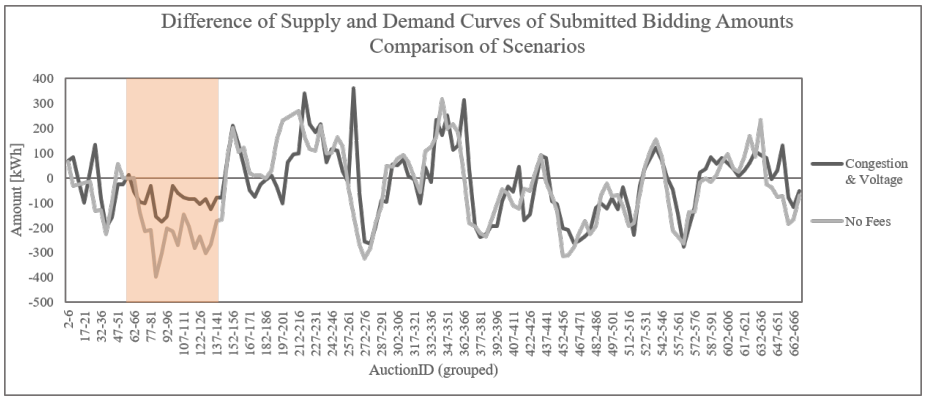


Figure 6.10 – Difference of Supply and Demand Curves of Submitted Bidding Amounts - Comparison of Scenarios

However, it is unsure what causes the demand to be higher in Scenario 1, because SOC and voltage was reviewed and no influence could be seen, hence, the agents should act identically for this period. In order to rule out that other influences affected this effect, the scenarios were redone with the same input data and without the previously mentioned effect of nodes synchronizing late. It has the same results; the MCP of Scenario 1 rises faster.

Agents use the voltage level as an indicator for the current supply and demand ratio in the market. When the voltage is high, consumers assume that the supply is also high and bid more progressively (bidding for more electricity at a lower price). Simultaneously, producers try to store more energy in their batteries, because selling it at the market would cause them to face additional fees. When the voltage is low, producers assume that the demand is high and bid more progressively (trying to sell more energy at a higher price). Simultaneously, consumers use as much energy as possible from their batteries, because buying at the LEM would, for them, lead to additional fees. This effect cannot directly be seen in the MCP, but it can be seen in the Figure 6.13.

Figure 6.11 and Figure 6.12 illustrate the different used bidding types for the different scenarios. Table 6.1 briefly explains the different implemented agent bidding types. In Figure 6.11, the bidding types are always the most conservative for producers (1) and the most conservative for consumers (4). In Figure 6.12, there exists an abundance of variation among the bidding types. Agents often bid more progressively, based on the voltage information they acquire. It can be seen that, during times of high voltage (e.g. AuctionID 187-212), consumers bid more

Number	Explanation
0	Random bid between 8.0 and 26.0 cents.
1	Conservative producer bid (0.2 to 0.6 cents under MCP)
2	Very conservative producer bid for voltage peaks (0.6 to 1.0 cents under MCP)
3	Progressive producer bid for voltage lows (0.0 cents to 0.4 cents over MCP)
4	Conservative consumer bid (0.0 to 0.4 cents over MCP)
5	Progressive consumer bid for voltage peaks (0.0 to 0.4 cents under MCP)
6	Very conservative consumer bid for voltage lows (0.4 to 0.8 cents over MCP)

Table 6.1 – Agent Bidding Types

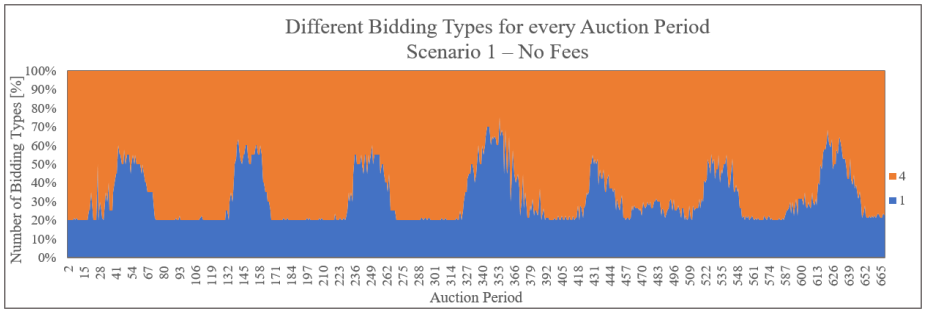


Figure 6.11 – Different Bidding Types for every Auction Period - Scenario 1

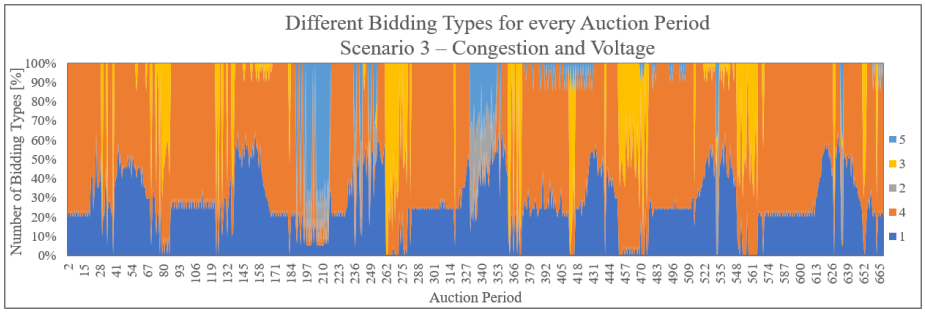


Figure 6.12 – Different Bidding Types for every Auction Period - Scenario 3

progressively (bidding type 5 in Figure 6.12 instead of 4 in Figure 6.11). During times of low voltage (e.g. AuctionID 250-280), producers bid more progressively (bidding type 3 in Figure 6.12 instead of 1 in Figure 6.11).

In order to be able to bid differently, the agents must be flexible and use their batteries. The difference in the battery usage can be seen in Figure 6.13. An example where the batteries are used grid-friendly, can be seen between AuctionID 187 and 212. Figure 6.8 shows here a deviation of more than 3 V from 230 V, that is when

intelligent agents start to act against the voltage deviation. In Figure 6.13 can be seen that the SOC in the "Congestion & Voltage" scenario rises here, whereas the SOC of the "No Fees" scenario stays constant. Later in the digram, more deviations between the two scenarios can be seen, which mainly depend on the agents reaction to the voltage deviation. The fact that the battery SOC is mostly at the lower SOC band threshold also shows that the given flexibilities (batteries) are still too little. More flexibilities would allow a even better contribution of the market to the grids stability, in this specific case, the agents could interfere better when the voltage is too low.

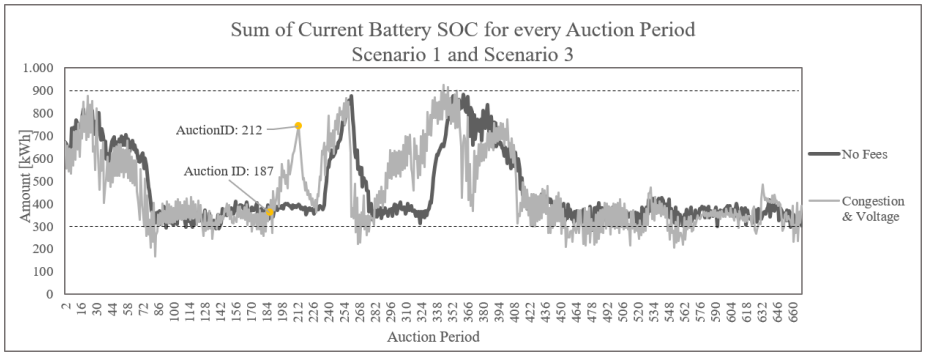


Figure 6.13 – Sum of Current Battery SOC for every Auction Period - Scenario 1 & 3

Figure 6.14 and Figure 6.15 show the balances of the congestion and voltage accounts for Scenario 3. Figure 6.14 also shows the amount that was procured outside of the LEM (the difference between total amount produced and total amount consumed for one auction period) to better illustrate how the congestion account balance rises more quickly when a lot of electricity is procured outside of the LEM. The voltage account balance only sinks during times of high voltage, but never during times of low voltage and it never rises. This indicates a bug in the GridFee or DoubleAuction smart contract.

Scenario 1 and the figures presented prove that the agents, as well as the market, react quickly to the current supply and demand. In the comparison between Scenario 1 and the other scenarios it can be seen that the agents use their given flexibilities in order to react (change bidding type and submitted amount) to different voltage levels and contribute to the markets stability. They react this way, because they want to avoid the additional voltage fees that are implemented. Frequency fees and congestion fees are also implemented in the market, but the agent implementation has to become more intelligent and versatile in order to be able to react to these implemented fees.

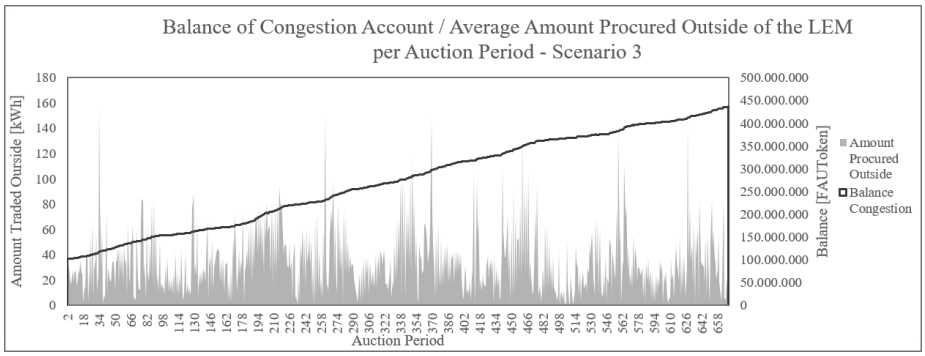


Figure 6.14 – Balance of Congestion Account / Average Amount Traded Outside of the LEM per Auction Period - Scenario 3

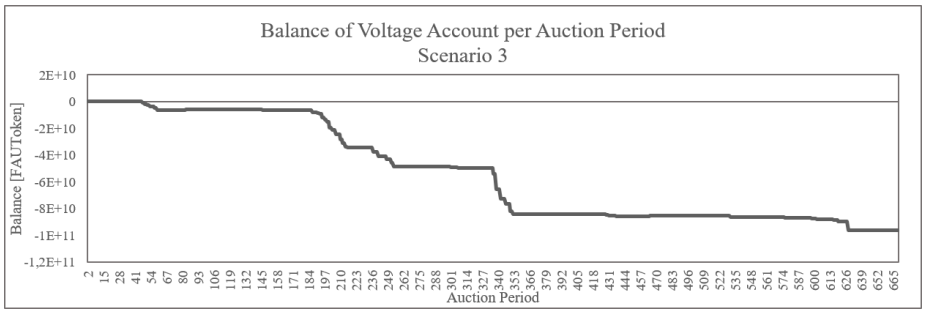


Figure 6.15 – Balance of Voltage Account per Auction Period - Scenario 3

6.1.3 Comparison of Semi-Autark to Autark Approach

This section focuses on comparing the autark voltage fee approach to the semi-autark voltage fee approach. To do this, the defining of two scenarios and conducting of multiple simulations took place. The first is the same scenario as Scenario 3 in the previous section. It covers 670 auction periods and congestion, as well as the semi-autark voltage fee approach is used. The second scenario for this comparison is similar, but uses the autark voltage fee approach.

The MCP and battery SOCs of the agents are not affected by switching between different approaches, because the agents do not differ between the autark and the semi-autark approach; they always act the same. The only thing that differs is the voltage account balance, as seen in Figure 6.16.

Figure 6.16 illustrates that the account balance for the semi-autark approach varies, while the balance of the autark approach remains constant. This happens by design and has many benefits because, this way, the market does not rely on money that comes from the outside.

The two approaches were only discussed briefly because the current differences having to do with the market are quite minor, considering agents do not actively

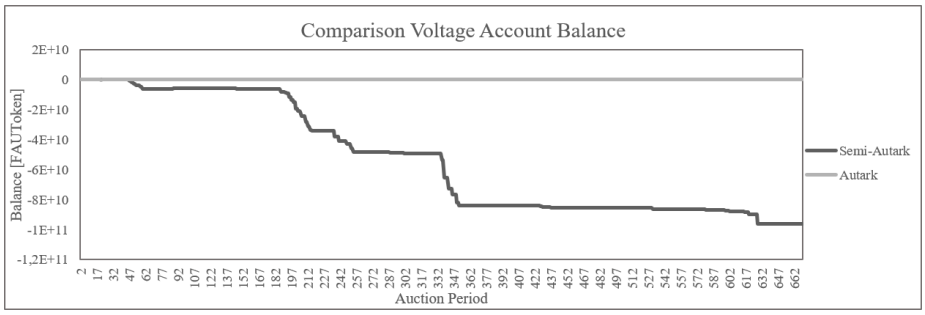


Figure 6.16 – Comarison of Balance of Voltage Account per Auction Period

react yet to different approaches or different fee amounts (which can be the result of different approaches).

6.1.4 Comparison of Intelligent to Zero-Intelligence Agents

This section compares the ZI with intelligent bidding agents. In both scenarios, the same input data is used, but the ZI agents send random bids in a predefined price range, whereas the intelligent bidding agents act based on the current MCP, voltage, and battery SOC.

Figure 6.17 shows the MCP of the intelligent and ZI agent scenarios.

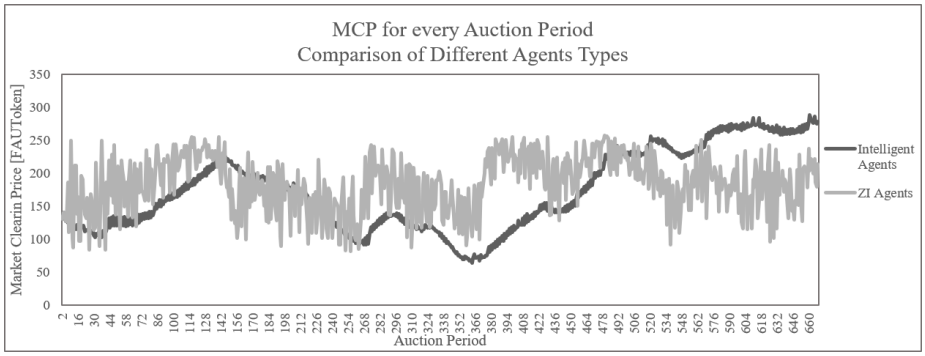


Figure 6.17 – MCP for every Auction Period - Comparison of Different Agents Types

As shown in Figure 6.17, the MCPs vary significantly. The MCP in the ZI agent scenario can jump from 10.0 cents in one auction period, to over 20.0 cents in the following, whereas the MCP of the intelligent agent scenario barely changes more than 1.0 cent between two auction periods. Even though the MCP in the ZI agent scenario seems fairly random, it still shows trends similar to the MCP in the intelligent agent scenario. This is shown clearly between auction period 2 and 130, where both trends face upward, as well as between auction period 130 and 260, where both trends face downward. This occurs because the ZI agent bids are still bound to fixed input data and the given difference between supply and demand always leads the price in the corresponding direction.

When it comes to battery usage, the two scenarios do not differ greatly because the battery usage is mainly based on the supply/demand input and voltage. The only reason why the curves vary slightly is because the MCPs differ. But all in all, the two curves in Figure 6.18 follow a similar pattern.

The analysis of the amount that has been traded within (success at double auction) or outside the LEM (with the public grid) proves to be quite interesting, and is visualized in Figure 6.19. Figure 6.19 shows that all the scenarios using intelligent agents, trade fairly large amounts within the LEM (between 62 and 69%), whereas in the scenario with ZI agents, only 44% where traded within the LEM. This is because the intelligent agents bid based on the last MCP (and other information they gathered) and try to use this information to submit a bid that has a very high chance

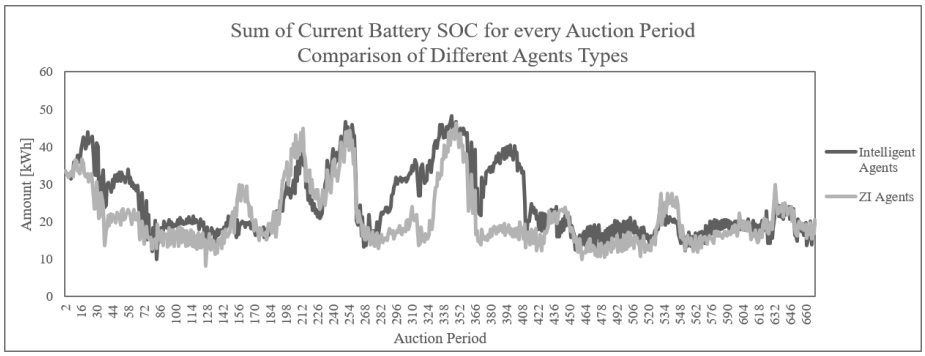


Figure 6.18 – Sum of Current Battery SOC for every Auction Period - Comparison of Different Agents Types

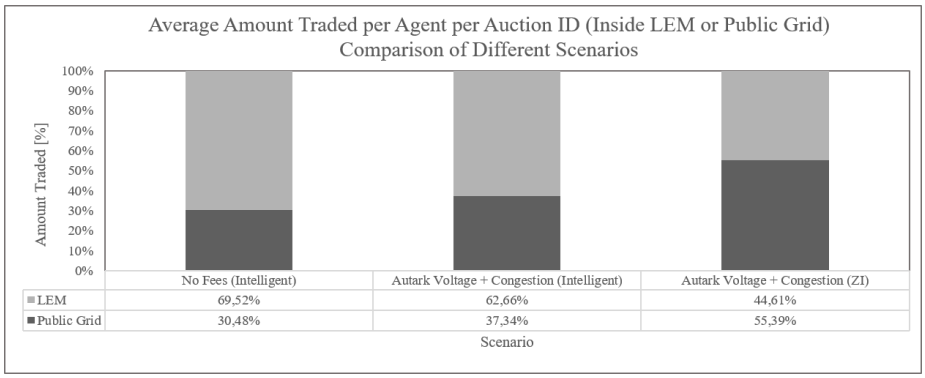


Figure 6.19 – Average Amount Traded per Agent per Auction ID (Inside LEM or Public Grid) - Comparison of Different Scenarios

of being successful at the LEM. They do this because being unsuccessful can lead to additional fees (congestion) and higher prices (at the public grid). ZI agents, on the other hand, do not use any data in order to estimate the different bids' chance of success. As a result, intelligent bidding agents are more successful at the LEM than ZI agents.

This section highlighted the differences and similarities between intelligent and ZI agents. It is evident that intelligent agents perform better at the LEM, even though the implemented intelligence is quite rudimentary. Agents that are more intelligent and flexible, who gather even more data about the market and have access to more flexibilities (batteries), may perform better. A better agent performance also leads to a better market performance where more participants are successful at the market.

6.1.5 Market Simulation Discussion

This section focused on the market, its agents, and the differences between the various scenarios conducted.

It became clear that the agents play a vital role in the overall performance of the market. The implemented intelligent agents performed well, but they did not react to every feature that the market has implemented. Hence, agents should gather more data, especially about the grid's frequency and the congestion level, and then utilize this data to react to changes in the frequency or congestion level. Furthermore, they should be able to react to different fee rates because fee rates can change too. For example, if the approaches (autark/semi-autark) change, fee rates change as well, because the autark approach only bounds the fines to fixed fee rates, making the rewards then based on the amount collected from the fines.

Another important aspect of the agents has to do with their synchronization with the blockchain. During the simulations, we ran into the problem that three agents/nodes constantly were not submitting their bids in time. This is crucial for a successful comparison, but also for a successful market, where all participants have the same chances of success. Since the problem did not appear during the first conducted scenario (the scenario with no fees), we made the assumption that the three slower nodes experienced some kind of hardware issues (probably RAM or disk space). This issue should be analyzed in more detail.

The voltage grid fee's previously mentioned bug should also be analyzed in more detail and may be connected to another problem that appeared during the analysis of the data. The participants' balances, for the most part, only rise and never sink. When they do manage to sink, it's only by a comparatively very low level. This indicates that a malfunction exists in either the escrow or one of the transfer functions.

In addition to the two mentioned balance bugs, another programming mistake appeared, in which the amount of the bids was sent in Wh and directly became multiplied with the MCP, even though the MCP is based on kWh, not Wh. This led to a miscalculation of the amount to be paid by a factor of 1000.

Another market issue that did not appear, but should be mentioned, is the fact that auction period lengths can vary based on the timing of submitted bids. For our scenarios, every auction period length equaled the goal of 30 seconds. The issue is only the case when agents submit bids immediately after the end of the previous auction period. In the case that no agent sends bids for, lets say 42 seconds, the auction period length will automatically be 45 seconds long (that's when the next block is mined). This can lead to problems because the agents submit their bids based on fixed time periods (usually 30 seconds).

The markets stability can further be improved by combining the bid and actualUsage function in one transaction call. This would lead to less traffic in the network, because less transactions are executed and it would prevent bugs that occur when agents submit bids but no usage (for example when the bid transaction gets

accepted, but the actualUsage transaction reaches the corresponding auction period too late, because the node had synchronization issues).

In order to let the market scale better, some of the information that is currently processed on-chain, should be considered to be processed off-chain. For example, information about the grids status should be reviewed and checked if it makes sense to spend gas for the corresponding transactions and store this information redundantly on the blockchain.

Another possible market improvement, that should be mentioned, is connected to the allocation of the gas costs. Section 6.2.4 shows that the marketClear and escrow functions are by far the most gas intensive transactions. The first agent that submits a bid to a new auction period has to call the marketClear and escrow function. Consequently, this agent also has to pay for the incurring gas costs. It would be fatal for the market if agents would try to submit their bids as late as possible just to avoid paying for the marketClear and escrow functions, because this could lead to fluctuating auction period lengths, as mentioned earlier. In order to avoid this issue, the agent that calls these gas intensive functions should be reimbursed.

What proves problematic is the fact that every transaction can be viewed by other participants, or even in general by anyone who has access to the blockchain. This essentially means that no account balance or traded energy amount is private.

Nevertheless, the simulation of the implemented market and its agents show that a blockchain-based LEM can indeed be implemented and also contribute to grid functions. The purpose behind this implementation was to function as a prototype, which it most definitely achieved.

6.2 Blockchain Performance

This section focuses on the performance of the i7Chain. The number of validator nodes, the impact of the blockchain-depth, the resource usage, and the scalability, will be reviewed and discussed.

6.2.1 Number of Validator Nodes

Ideally, every node of the i7Chain should have a validator/miner. This would give every participant the same power and the system would be completely decentralized. Unfortunately, a higher number of miners destabilizes the network because it causes forks to occur more often. Moreover, every node would have to be a "full" node and has to store more data in comparison to "light" or "fast" nodes. These node types are further explained in the geth documentation [65].

For testing purposes, the i7Chain was initialized with a different number of miners, but always 20 nodes in total. It was initialized with 1, 3, and 20 miners,

and ran each time for 120 blocks. The result was that no forks appeared with 1 or 3 miners, but many appeared when 20 miners were used. Figure 6.20 illustrates the number of side forks that occurred for each node.

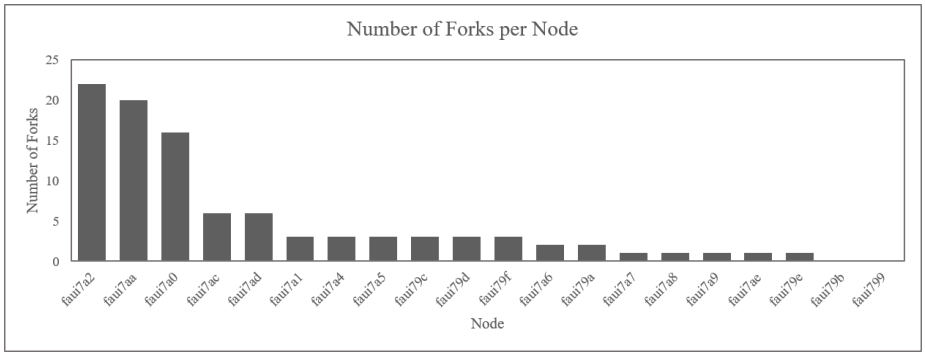


Figure 6.20 – Number of Forks per Node

It is clear in Figure 6.20 that the nodes faui7a2, faui7aa, and faui7a0 have the most trouble in synchronizing with the main chain. In comparison to the other nodes, these three nodes appear to need more time for synchronization because the agents of these nodes often have trouble to send their bids in time.

6.2.2 Impact of Blockchain-Depth

This section reviews the hypothesis stating that with growing blockchain-depth, the gas costs for transactions rise as well. This hypothesis is based on the thought that the longer its chain becomes, the deeper the blockchain has to dig in its blocks, which could lead to additional computation/gas costs. To explore this hypothesis, one market scenario with 670 auction periods and over 3000 blocks is analyzed. Figure 6.21 shows the gas costs for the bid function, without the marketClear and escrow call, and the actualUsage function. It is visible that both functions vary a little, but all in all, they are quite consistent and do not show any kind of trend. Figure 6.22 illustrates the gas costs for the marketClear and escrow functions. These costs vary significantly due to the bidding rates, but again, no trend is indicated.

Consequently, no impact from the blockchain depth on gas costs can be proven.

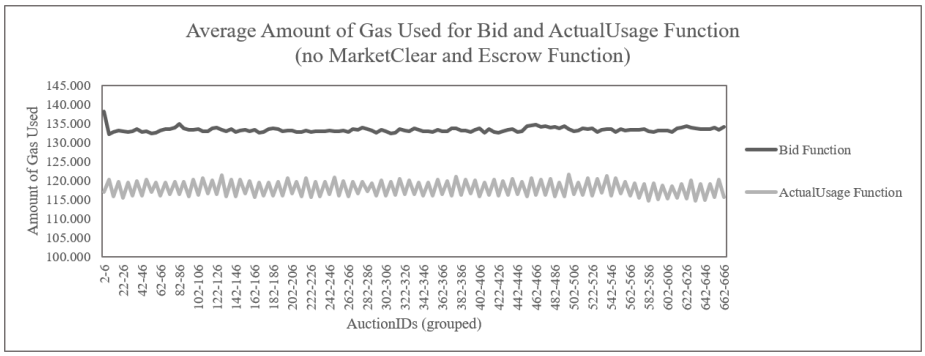


Figure 6.21 – Average Amount of Gas Used For Bid and ActualUsage Function (No MarketClear and Escrow Function)

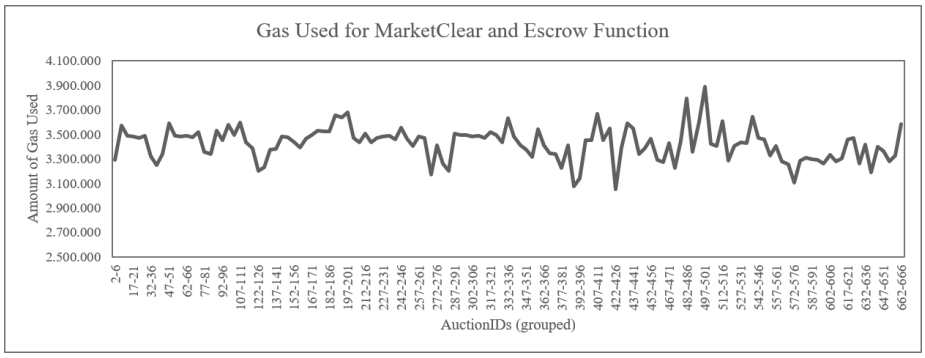


Figure 6.22 – Gas Used For MarketClear and Escrow Function

6.2.3 Resources

In this section, the blockchains use of different available resources (such as CPU usage and storage space) is reviewed.

As explained in Section 5.2, every node of the i7Chain has approximately 350 MHz of computing power, divided into two cores and 4096 Mb of RAM. A python script, called CPU_Usage, is used to gather as much information as possible about the resource use of the i7Chain's two nodes, faui799 and faui79a. Node faui799 is the only authority node in the network for the reviewed simulation, hence, it has to validate every block. The script uses the psutil library and collects information about the total CPU usage, the CPU usage of the geth process (via pid), and the CPU usage of the startAgent python process (via pid) all in percent form, as well as some additional data about the RAM usage, disk reads and writes, and network traffic. The additional data does not show any peculiarities.

However, the CPU data does show interesting patterns. Figure 6.23 displays the CPU usage of the geth process while no market was running. The vertical lines indicate the block timestamps. The figure shows that the usage goes up to almost

200% approximately every two minutes for each node. A value of more than 100% is only possible when the process runs on multiple threads, on different cores. The root of these peaks is not analyzed in more detail, but this data is used as a reference for further analysis. Figure 6.23 also illustrates that, besides the 200% peaks, the usage barely exceeds 20%.

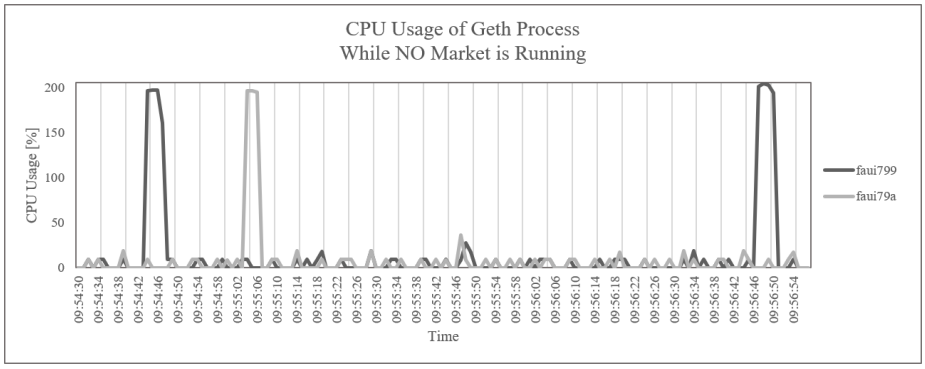


Figure 6.23 – CPU Usage of Geth Process While No Market Is Running

Figure 6.24 shows what the CPU usage of the geth process for both nodes was while a market was running. The thicker, vertical lines represent the timestamps of the last blocks in each auction period. The 200% peaks are visible again, but what's more interesting are the peaks that occur at the start of every new auction period. These peaks reach up to 100% and indicate that the node does not have sufficient processing power for finishing the outstanding computations in time. The figure also shows that faui799, the validator node, is busier during the first block of an auction period.

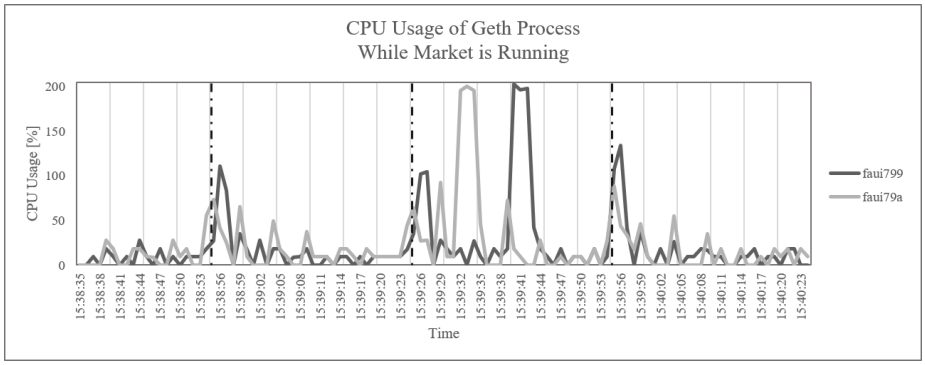


Figure 6.24 – CPU Usage of Geth Process While Market Is Running

In addition to the CPU usage, the storage usage was also reviewed in more detail. The size of the relevant folder, the chaindata folder, was checked, the size stored, and then checked once more after 300 blocks were mined. This procedure was first

conducted without a running market, and then with a running market containing 20 participants. Figure 6.25 shows the results per block.

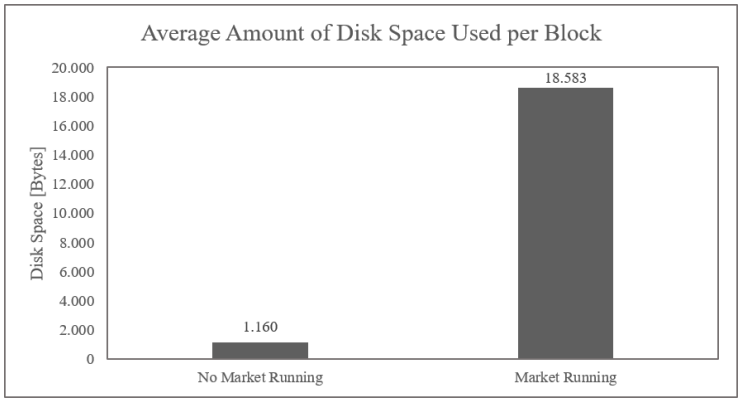


Figure 6.25 – Average Amount of Disk Space Used Per Block

It is clear that the running market consumes much more disk storage per block. While the market is running, a block requires almost 20 KB, on average. Based on these circumstances, Equation (6.1) can be formulated, where t is the time period in seconds in which the market with 20 participants is active with 30-second auction periods and 5-second block times. $Block_{\text{Number}}$ is number of blocks per auction period, $AuctionPeriodLength$ is the length of an auction period in seconds and $BlockSize_{\text{chaindata}}$ is the space per block that was required for the market with 20 participants in Kilobytes. The result is in Kilobytes.

$$DiskSpace = \frac{t}{AuctionPeriodLength} \cdot Block_{\text{Number}} \cdot BlockSize_{\text{chaindata}} \tag{6.1}$$

Consequently, a market with 20 participants that runs for 30 days would consume about 10 GB of disk space on every node. This makes a consumption of 200 GB in total, where 190 GB consists of redundant data.

6.2.4 Scalability

Scalability refers to the extent in which a network or process is able to be enlarged, while simultaneously continuing to accommodate its own growth. The i7Chain has a maximum of 20 nodes. In order to know whether the implemented LEM is able to scale or not, different tests with varying numbers of nodes are conducted and reviewed.

As explained in Section 5.2, the auction period time for the market is set to 30 seconds and the block time is set to 5 seconds. This is mainly because the market should clear in periods that are as short as possible (in order to be able to offer grid-stabilizing measures), but also because there should be about six blocks per

auction period to both prevent forks and minimize the chances of agents missing an auction period. These numbers are only valid for about 20 nodes.

In order to gain a better understanding of these numbers, a test with 5, 10, and 20 random ZI agents was conducted. Each case was set up with a newly deployed market and agents who participated in at least 16 auction periods. Afterwards, the ChainListener script processed all of the blocks and stored any relevant information gathered.Figure 6.26 shows the average number of transactions per block for each case.

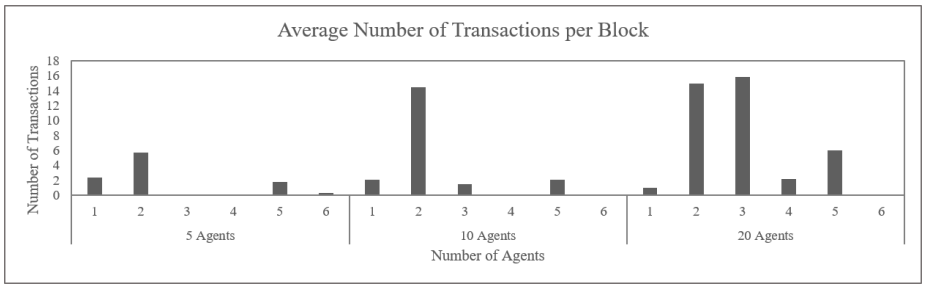


Figure 6.26 – Average Number of Transaction Per Block

In the case of 5 agents, it is a total of 10 transactions per auction period (5 times sendBid and 5 times sendUsage), in the case of 10 agents, it is a total of 20 transactions, and in the case of 20 agents, it is a total of 40 transactions. Figure 6.26 illustrates that when only 5 or 10 agents participated, many blocks were empty. In the case of 20 agents, on the other hand, only one block stays empty. This happens by design. We chose a 30-second auction period length for 20 agents because it is the shortest possible period in which we still have one block as a cushion. On average, the fifth block has more transaction than the fourth block because some nodes in the network require more time to synchronize with the blockchain and, consequently, submit their transactions a little later.

The first block in an auction period always contains the marketClear and escrow functions, both of which consume a lot of gas. For this reason, the first block usually uses a lot of gas (as seen in Figure 6.27), but doesn't contain many transactions. In the case of 5 agents, it can contain up to 6 transactions, but in the case of 10 agents, it can only contain up to 4 transactions, and in the case of 20 agents, it never contains more than 1 transaction. This leads to the conclusion that the gas costs of these functions rise with an increasing number of participants.

Figure 6.27 also shows that at about 2,200,000 gas, no more transactions are added to the block. The gas limit is set to 20,000,000, meaning that it is not the reason behind the stopped transactions. However, as seen in Section 6.2.3, the CPU load could be responsible for the limit at about 2,200,000 gas.

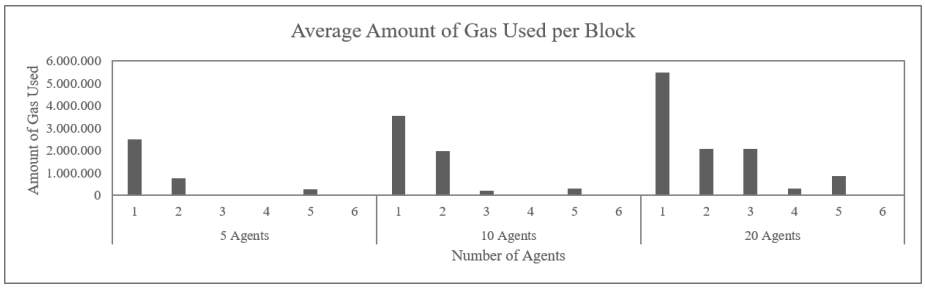


Figure 6.27 – Average Amount of Gas Used Per Block

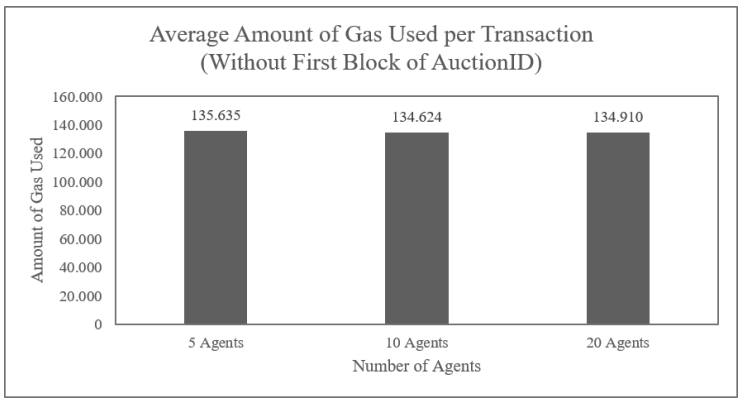


Figure 6.28 – Average Amount of Gas Used Per AuctionID Per Agent Without First Block

Figure 6.28 indicates that an agent usually spends about 135,000 gas per auction period, per transaction, and this amount does not depend on the number of agents that are participating. The calculation for this figure does not include the first block of an auction period because this block would also contain the gas used by the marketClear and escrow functions, which will be analyzed in more detail later. Based on the 135,000 gas per transaction, every agent spends at least 270,000 gas per auction period, excluding the marketClear and escrow costs.

Figure 6.29 illustrates the average amount of gas that is spent on the marketClear and escrow functions per AuctionID, as well as per agent per AuctionID. It shows that the total gas costs of these functions depend on the number of participating agents. It also shows that the costs per agent become less with a growing number of agents.

Figure 6.30 shows that the gas costs for the marketClear and escrow functions vary significantly, even when the same amount of agents are participating. The difference between the minimum and maximum amount can be by a factor 2. This is a result of the different bids that were sent to the market. If the MinRate and

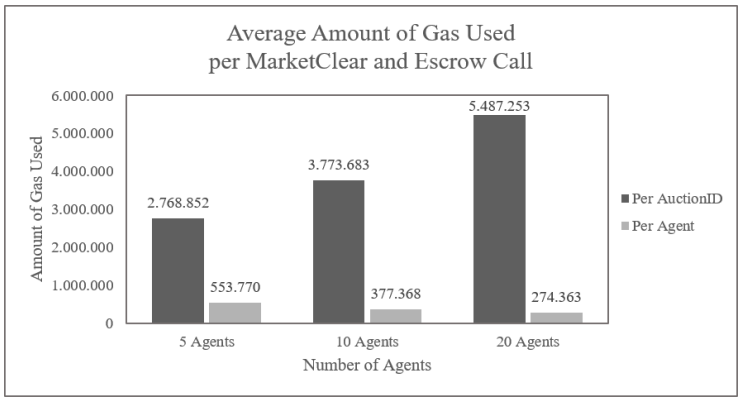


Figure 6.29 – Average Amount of Gas Used Per MarketClear and Escrow Call

MaxRate value of the auction differ greatly from one another, the for-loop in the marketClear function will be longer, which will cause the gas costs to rise.

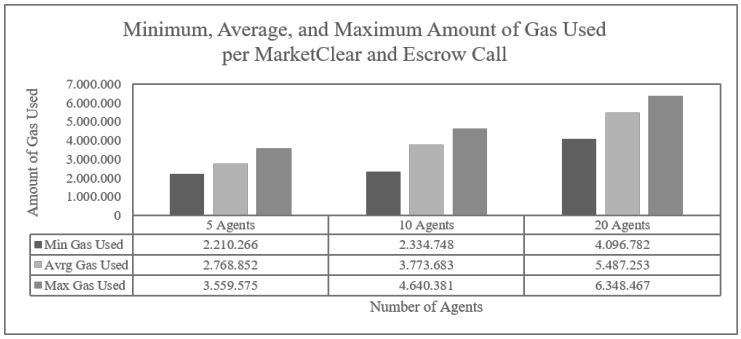


Figure 6.30 – Minimum, Average, and Maximum Amount of Gas Used Per MarketClear and Escrow Call

The above-mentioned facts lead to the following conclusion:

- If more than 20 agents are participating at the market, the first block of every auction period will only contain one transaction, which is mainly used for the marketClear and escrow functions.
- All other blocks are limited to 2,200,000 gas per block (at least with the current hardware settings).
- Every agent spends 270,000 gas per auction period on bidding and submitting the actual usage (without MarketClear and Escrow functions).
- A block can contain the transactions (here bidding and actualUsage are assumed to be one transaction) of up to 8 agents, which can be calculated with

Equation (6.2), where $MaxGas_{\text{perBlock}}$ equals 2,200,000.

$$TX_{\text{perBlock}} = \lfloor MaxGas_{\text{perBlock}}/270,000 \rfloor \tag{6.2}$$

- If n is the number of participating nodes/agents and is greater than or equal to 20, and if the block gas limit is sufficient for calling the escrow and marketClear function, then the number of necessary blocks per auction period can be calculated as in Equation (6.3). The number 3 represents three blocks; one block that contains the marketClear and escrow functions, one block for nodes that synchronize late, and one block as cushion:

$$Block_{\text{Number}} = 3 + \lceil n/TXs_{\text{perBlock}} \rceil \tag{6.3}$$

- Based on both Equation (6.3) and the fact that block times should not be shorter than 3 to 5 seconds, as explained in Section 5.2, the auction period can be calculated as in Equation (6.4). A block time of 5 seconds is assumed. The result is in seconds.

$$AuctionPeriodLength = Block_{\text{Number}} \cdot 5s \tag{6.4}$$

- Based on the gas that was used for the marketClear and escrow functions in the conducted tests, as seen in Figure 6.29, Equation (6.5) can be formulated. The underlying data only contains three datapoints, hence, Equation (6.5) is based on a rough estimation. A linear function is assumed to represent the block gas limit, which is necessary to call the marketClear and escrow functions, depending on the number of participating agents. The first part of the equation is multiplied by 2 because it only refers to the average amount of gas used, but as illustrated in Figure 6.30, the necessary amounts vary.

$$BlockGasLimit = (2,200,000 + 150,000 \cdot n) \cdot 2 \tag{6.5}$$

  However, the block gas limit should never be less than 9,500,000, because this is the amount necessary to deploy the DoubleAuction contract. The deployment of the other smart contracts is significantly cheaper.

The listed points show that the scalability of the market is limited. The equations introduced easily allow one to estimate the necessary key elements. For example, a market with 100 agents and the same blockchain settings would need at least 16 blocks per auction period, according to Equation (6.3). Therefore, the auction period lasts 80 seconds, based on Equation (6.4), and the block gas limit needs to be higher than 34,400,000, according to Equation (6.5).

6.2.5 Blockchain Performance Discussion

This section pointed out the main performance issues having to do with the i7Chain. All the information is only valid for the current blockchain and hardware settings. For example, the forks that increasingly occur with a higher number of validators are completely dependent on the consensus mechanism used. For the i7Chain, the consensus mechanism used is clique. Other PoA mechanisms, such as aura, potentially perform better, with fewer side forks occuring. However, as long as clique is used, it is recommended to keep the number of miners to a minimum.

Section 6.2.3 focuses on the hardware settings of the i7Chain. It pointed out that the given CPU is probably a bottleneck for the mining of transactions. In order to understand this better and have proof to justify it, the i7Chain should be tested again with different hardware settings. In regard to the storage usage of the i7Chain, Equation (6.1) was introduced. This equation calculates the necessary disk space for "full" Ethereum nodes. As briefly mentioned in Section 6.2.1, less storage-intensive nodes are also possible. If only a small amount of miners is used anyways, the remaining nodes can run as "light" or "fast" Ethereum nodes. The use of these nodes has the potential to decrease storage utilization, leading to less redundant data.

If a better CPU would result in a potential of more transactions per block, the $MaxGas_{\text{perBlock}}$ variable would change as well. Consequently, fewer blocks would be necessary per auction period (based on Equation (6.3)) and auction periods would have the potential to be shorter (based on Equation (6.4)). Overall, this would lead to a better scalability of the market itself, supporting that the market should be tested with better hardware settings.

Another possible improvement that was not yet mentioned is the optimization of smart contracts in regard to gas usage. The implemented smart contracts mainly focus on the functionality of the market. They are implemented as a prototype and bear a high potential to minimize gas costs. If decreasing the different solidity functions' gas costs is possible, more transactions will be able to fit in the same block, even with the current actual block gas limit set at 2,200,000. The 270,000 constant in Equation (6.2) would become smaller, while the result of the equation would become bigger.

The current i7Chain over-performs blockchain simulation tools, such as Ganache, which has an even lower actual block gas limit, a very high storage usage, and troubles with too many transaction calls occuring at the same time. Unfortunately, the average block on EWF's testnet, Tobalaba, uses only about 300,000 gas, making it hard to estimate how this network would perform when running more computing-intensive transactions, such as the LEM's transactions implemented in this thesis. With that said, it would be interesting to see how the implemented smart contracts would perform on Tobalaba.

All in all, many possibilities of further improving the i7Chain were presented in this section, making the potential of a more stable and scalable blockchain-based LEM easier to envision.

Chapter 7

Conclusion

The switch to more renewable energy is inevitable. However, with the rise of renewable energy, the electricity markets must also be significantly reformed. The current approach is to produce and consume the same amount of electricity in a given area (LEM), in order to avoid any unnecessary transmission of electricity.

To make this happen, the price establishment of electricity must be much more flexible than it is today; with more flexibility, even small market participants will have a chance to profit from the grid's current supply and demand (by buying at a low rate during times of high consumption and selling at a high rate during times of high demand). In addition to a more flexible price establishment, another idea to reform the electricity market's complete infrastructure is to permit market participants to assist in sustaining the grid's stability. This is possible through the introduction of action-based, real-time grid fees, which are applied to every Wh that is bought or sold, based on the current grid state.

In this thesis, a variety of electricity markets were implemented, all having the same core functions: the offering of supply and demand based prices to small- and medium-sized market participants, as well as, the charging and rewarding of grid-influencing measures. Both happening close to real-time. These markets were implemented on a private and permissioned Ethereum instance that performs as information layer and is referred to as the i7Chain. The idea of using blockchain technologies for applications like this is fairly new. It has not been researched yet for this particular use-case and, thus, leads to many challenges. However, this unique usage also offers many advantages, such as higher transparency, better security, an ability to fit the decentralized nature of energy production/consumption, and no necessity for intermediaries. Additionally, agents were implemented to react to the markets and represent market participants for a variety of conducted simulations.

The simulations showed that the market is able to react to supply and demand, offer real-time prices, and assist in sustaining the grid's stability. It was also shown

that the agents used their flexibilities if they had some in the form of batteries, which helped to flatten supply and demand curves and, ultimately, support the grid's stability. The markets were implemented as prototypes and proved that they can fulfill their functions.

However, while the simulations were conducted, many improvements were found that carry the potential to lead to a more stable and fair market interaction. One of the most essential aspects is the fairness of the market, which includes that every participant pays and receives the correct amount of money. The current implementation did not yet fully incorporate this. Furthermore, aside from making the market transparent, the implementation on Ethereum also makes it insecure in terms of private data, such as account balances or traded energy amounts. This issue can be resolved by switching to different blockchain technologies (e.g. Hyperledger Fabric).

Another important point that came up is that in order to be able to fully review the market, more intelligent agents must be implemented. Agents that are not only able to react to every little change at the market, but also to the different implemented grid fees. In order to do that, they must collect more information about the market, the grid's state, and their own electricity usage profiles, and process this information accordingly.

The i7Chain used for the conducted simulations performed well. However, many improvements can be made in terms of hardware and the consensus mechanism to offer a fair, but stable network. The current market implementation on the i7Chain also does not scale well, but changes in hardware and a reduction in gas costs (through improving the used smart contracts) could improve this and lead to a more scalable market. Testing the implemented market on testnets, such as EWFs Tobalaba, could also deliver more insights.

In brief, the core of this thesis revolves around the conception of a grid-friendly and blockchain-based LEM with intelligent bidding agents. It presented a concept that was implemented as a functioning prototype in Solidity and Python 3. The concept was first tested on blockchain simulation tools, and later a private Ethereum instance, referred to as the i7Chain, was initialized to create more realistic simulations. A number of different scenarios were defined and conducted as simulations, which led to deeper understandings of the market, as well as i7Chain's performance.

This thesis showed that the implementation of grid-friendly and completely decentralized LEMs is possible, however, it also revealed limitations that must be considered. Provided that the mentioned improvements take place, blockchain-based, grid-friendly LEMs can be very useful for solving the upcoming challenges of the energy markets and, after all, contribute to the fight against climate change.

List of Acronyms

API	Application Programming Interface
BMG	Brooklyn Microgrid
CDA	Continuous Double Auction
CPU	Central Processing Unit
dApp	Decentralized App
DTDA	Discrete Time Double Auction
DA	Double Auction
DAG	Directed Acyclic Graphs
DAO	Decentralised Autonomous Organizations
DL	Distributed Ledger
DLT	Distributed Ledger Technologies
DSO	Distribution System Operator
EVM	Ethereum Virtual Machine
EWF	Energy Web Foundation
FCR	Frequency Containment Reserve
FRR	Frequency Restoration Reserve
geth	go-ethereum
GUI	Graphical User Interface
IR	Individual Rationality
IDE	Integrated Development Environment

LAMP	Landau Microgrid Project
LEM	Local Energy Market
MAS	Multi-Agent System
MCP	Market Clearing Price
NEIC	Nash Equilibrium Incentive Compatibility
NYSE	New York Stock Exchange
OTC	Over the Counter
PE	Pareto Efficiency
P2P	Peer-to-Peer
PBFT	Practical Byzantine Fault Tolerance
PoA	Proof of Authority
PoS	Proof of Stake
PoW	Proof of Work
PV	Photovoltaic
RES	Renewable Energy Ressource
SOC	State of Charge
SPS	Stuff per Second
TPS	Transactions per Second
TSO	Transmission System Operator
VCG	Vickrey Clarke Groves
VM	Virtual Machine
WBB	Weak Balanced Budget
ZI	Zero Intelligence

List of Figures

List of Tables

Appendices

Appendix A

Oli System Smart Contracts

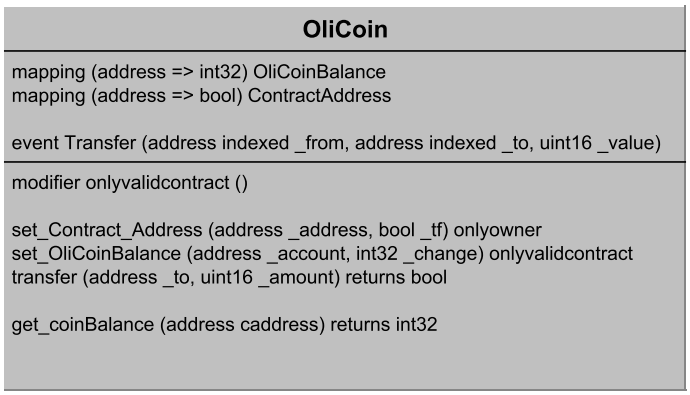


Figure A.1 – Oli System Smart Contracts OliCoin.sol based on [54]

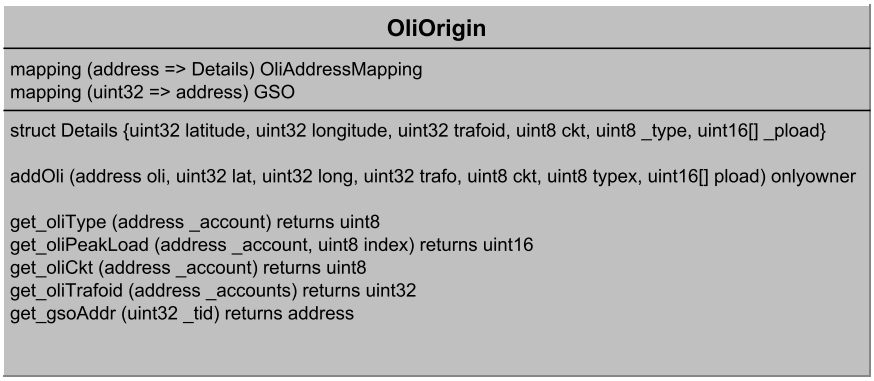


Figure A.2 – Oli System Smart Contracts OliOrigin.sol based on [54]

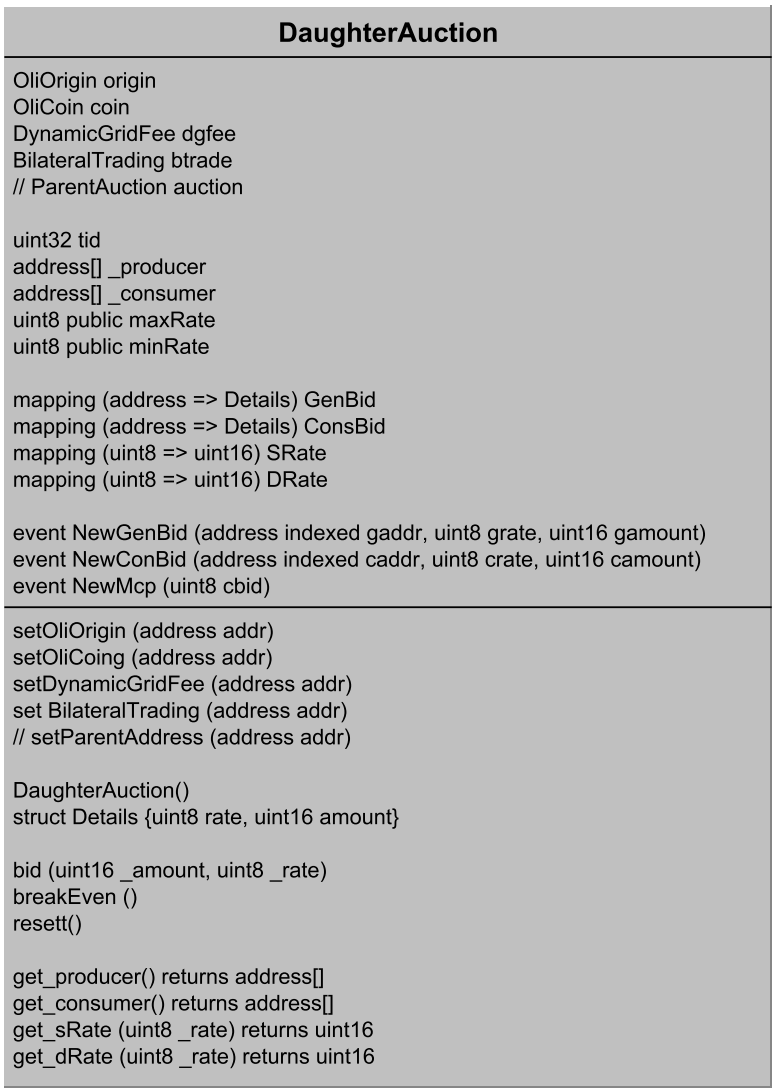


Figure A.3 – Oli System Smart Contracts DaughterAuction.sol based on [54]

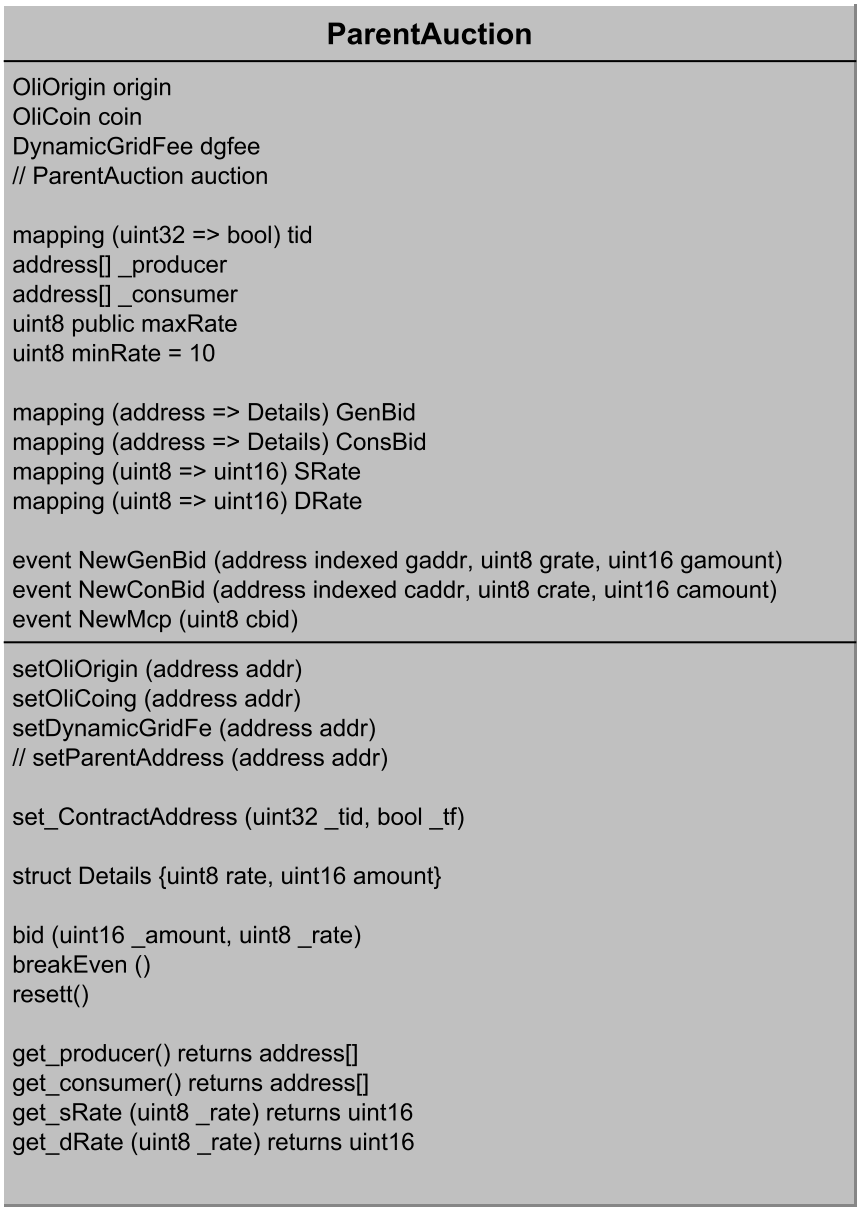


Figure A.4 – Oli System Smart Contracts ParentAuction.sol based on [54]

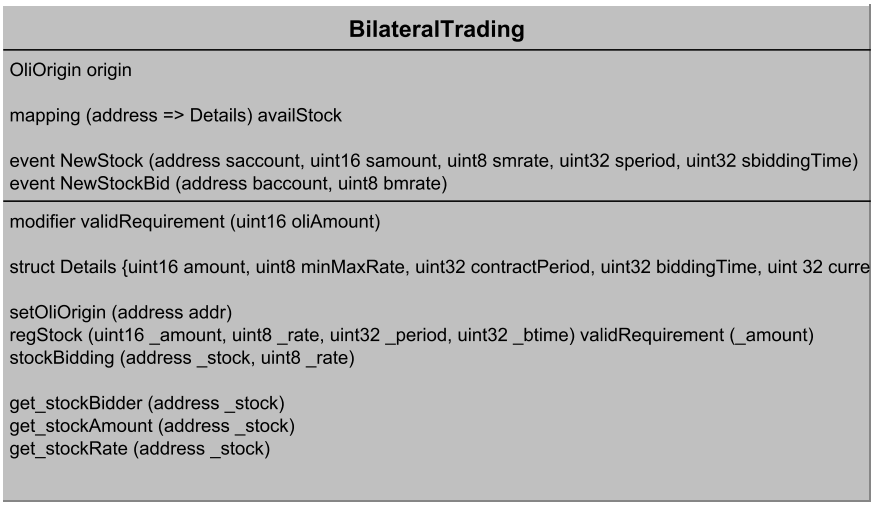


Figure A.5 – Oli System Smart Contracts BilateralTrading.sol based on [54]

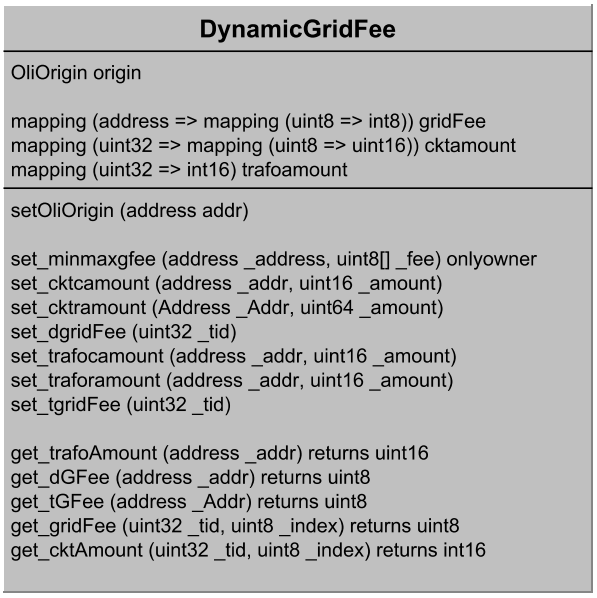


Figure A.6 – Oli System Smart Contracts DynamicGridFee.sol based on [54]

Appendix B

Digital Files

1: Literature

2: Implementation

- Python Scripts
- Smart Contracts

3: Analysis

- Blockchain Peformance
- Market Simulation
  - Simulation Data
  - Scenario Output Data

4: LaTeX Code and Input

5: Final Thesis

Bibliography

[1] WWF. (2018) The effects of climate change. [Online]. Available: https://www.wwf.org.uk/effectsofclimatechange

[2] Federal Ministry for Economic Affairs and Energy. (2018) Renewable energy. [Online]. Available: https://www.bmwi.de/Redaktion/EN/Dossier/renewable-energy.html

[3] C. Morris. (2018, Jan.) Germany lifts 2030 renewable energy target to 65[Online]. Available: https://reneweconomy.com.au/germany-lifts-2030-renewable-energy-target-65-12576/

[4] Council of European Energy Regulators, "Report on power losses," CEER, Tech. Rep., 2017.

[5] K. Kok, "The power matcher: Smart coordination for the smart electricity grid," Ph.D. dissertation, Technical University of Denmark, 2013.

[6] R. Dargaville. (2016) Despite the hype, batteries aren't the cheapest way to store energy on the grid. [Online]. Available: http://theconversation.com/despite-the-hype-batteries-arent-the-cheapest-way-to-store-energy-on-the-grid

[7] StromMagazin. Die Liberalisierung der Energiemaerkte und die Folgen. [Online]. Available: https://www.strom-magazin.de/info/liberalisierung-der-energiemaerkte/

[8] D. R. Graeber, *Handel mit Strom aus erneuerbaren Energien*. Springer, 2014.

[9] D. C. von Dinther and P. R. Madlener, "Smart grid economics and information management," RWTH AAchen, KIT, EON Energy Research Center, 2013.

[10] D. Filzek and P. Ritter, "Marktbedingungen und Zugangsvoraussetzungen zum Strommarkt," Sep. 2011.

[11] EUETS. Frequency containment reserve (fcr). [Online]. Available: https://www.emissions-euets.com/internal-electricity-market-glossary/793-frequency-containment-reserve

[12] ——. Frequency restoration reserve (frr). [Online]. Available: https://www.emissions-euets.com/internal-electricity-market-glossary/794-frequency-restoration-reserve-frr

[13] J. N. Reekers, "Optimisation and analysis of a prototype for smart grid energy management based on blockchain," Master's thesis, HWI Hamburg, 2018.

[14] CircuitGlobe. Methods of voltage control in power system. [Online]. Available: https://circuitglobe.com/methods-of-voltage-control-in-power-system.html

[15] J. Petinrin and M. Shaaban, "Impact of renewable generation on voltage control in distribution systems," *Renewable and Sustainable Energy Reviews*, 2016.

[16] N. I. Yusoff, A. A. M. Zin, and A. B. Khairuddin, "Congestion management in power system: A review," 2017.

[17] EnergiewendeTeam. (2018, Mar.) The german electricity grid: notoriously swamped? [Online]. Available: https://energytransition.org/2018/03/the-german-electricity-grid-notoriously-swamped/

[18] K. Appunn. (2016, Feb.) Re-dispatch costs in the german power grid. [Online]. Available: https://www.cleanenergywire.org/factsheets/re-dispatch-costs-german-power-grid

[19] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Nov. 2008.

[20] ——. (2009, Jan.) Bitcoin v0.1 released. [Online]. Available: https://satoshi.nakamotoinstitute.org/emails/cryptography/16/#selection-7.0-7.21

[21] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982. [Online]. Available: http://doi.acm.org/10.1145/357172.357176

[22] V. Buterin, "Ethereum: A next generation smart contract & decentralized application platform," *Bitcoin Magazine*, 2014.

[23] Linux-Foundation. (2018, Jul.) Hyperledger fabric. [Online]. Available: https://hyperledger.org/projects/fabric

[24] CoinMarketCap. (2018, Jul.) Coinmarketcap. [Online]. Available: https://coinmarketcap.com

[25] EddyVM. (16, Jan.) Let's explore the main data structures of a blockchain system. [Online]. Available: https://dev.to/eddy_wm/lets-explore-the-main-data-structures-of-a-blockchain-system-1g6m

[26] J. Schlund, L. Ammon, and R. German, "ETHome: Open-source Blockchain Based Energy Community Controller," in *Proceedings of the Ninth International Conference on Future Energy Systems*, ser. e-Energy '18, New York, NY, USA, 2018, pp. 319–323. [Online]. Available: http://doi.acm.org/10.1145/3208903.3208929

[27] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *2017 IEEE International Congress on Big Data (BigData Congress)*, June 2017, pp. 557–564.

[28] B. Curran. (2018, Jul.) What is proof of authority consensus? staking your identity on the blockchain. [Online]. Available: https://blockonomi.com/proof-of-authority/

[29] Blockgeeks. (2008) Blockchain glossary: From a-z. [Online]. Available: https://blockgeeks.com/guides/blockchain-glossary-from-a-z/

[30] K. Hickson. (2018, Jan.) 101: Blockchain terminology. [Online]. Available: https://medium.com/my-blockchain-bible/101-blockchain-terminology-874f007c0270

[31] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger byzantium version," 2018.

[32] Etherchain. (2018, Sep.) Average block time of ethereum network. [Online]. Available: https://www.etherchain.org/charts/blockTime

[33] V. Buterin. (2014, Jul.) Toward a 12 second block time. Ethereum Blog. [Online]. Available: https://blog.ethereum.org/2014/07/11/toward-a-12-second-block-time/

[34] S. H. Kai Krmer and. (2018, May) When it comes to throughput, transactions per second is the wrong blockchain metric. EWF. [Online]. Available: https://energyweb.org/2018/05/10/

[35] Enerquire. (2018, Apr.) Blockchain in the energy sector is moving forward - key takeaways from the eventhorizon 2018. [Online]. Available: https://www.enerquire.com/blog/

[36] M. Richardson. (2017, Nov.) Ethereum vs hyperledger. [Online]. Available: https://blockchaintrainingalliance.com/blogs/news/ethereum-vs-hyperledger

[37] K. Rilee. (2015, Feb.) Understanding hyperledger fabric: Byzantine fault tolerance. [Online]. Available: https://medium.com/kokster/understanding-hyperledger-fabric-byzantine-fault-tolerance-cf106146ef43

[38] S. Logan. (2018, May) Ethereum roadmap explained. [Online]. Available: https://thecryptograph.net/ethereum-roadmap-explained/

[39] S. Rehman. (2018, Jan.) Demystifying ethereum private blockchain. [Online]. Available: https://medium.com/@s4saif.121/demystifying-ethereum-private-blockchain-74f78ddf76fb

[40] M. R. Blouin and R. Serrano, "A decentralized market with common values uncertainty: Non-steady states," *The Review of Economic Studies Limited*, 2000.

[41] E. Mengelkamp, P. Staudt, J. Garttner, and C. Weinhardt, "Trading on local energy markets: A comparison of market designs and bidding strategies," in *2017 14th International Conference on the European Energy Market (EEM)*, June 2017, pp. 1–6.

[42] M. Mihaylov, S. Jurado, K. Van Moffaert, N. Avellana, and A. Nowe, "Nrg-x-change a novel mechanism for trading of renewable energy in smart grids," pp. 101–106, 01 2014.

[43] P. Vytelingum, S. D. Ramchurn, T. Voice, A. Rogers, and N. R. Jennings, "Trading agents for the smart electricity grid," in *AAMAS*, 2010.

[44] S. Parsons, M. Marcinkiewicz, J. Niu, and S. Phelps, "Everything you wanted to know about double auctions, but were afraid to (bid or) ask," 01 2006.

[45] R. B. Myerson and M. A. Satterthwaite, "Efficient mechanisms for bilateral trading," *Journal of Economic Theory*, vol. 29, no. 2, pp. 265 – 281, 1983. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0022053183900480

[46] D. Ilic, P. Silva, S. Karnouskos, and M. Griesemer, "An energy market for trading electricity in smart grid neighbourhoods," pp. 1–6, 06 2012.

[47] Z. Tan and J. R. Gurd, "Market-based grid resource allocation using a stable continuous double auction," in *2007 8th IEEE/ACM International Conference on Grid Computing*, Sept 2007, pp. 283–290.

[48] E. Mengelkamp, J. Gaerttner, K. Rock, S. Kessler, L. Orsini, and C. Weinhardt, "Designing microgrid energy markets: A case study: The brooklyn microgrid," *Applied Energy*, vol. 210, pp. 870 – 880, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S030626191730805X

[49] J. Gaerttner, E. Mengelkamp, and C. Weinhardt, "Decentralizing energy systems through local energy markets: The lamp-project," 03 2018.

[50] S. H. Clearwater, *Market-Based Control*. WORLD SCIENTIFIC, 1996. [Online]. Available: https://www.worldscientific.com/doi/abs/10.1142/2741

[51] S. Lamparter, S. Becher, and J.-G. Fischer, "An agent-based market platform for smart grids." pp. 1689–1696, 01 2010.

[52] A. Hahn, R. Singh, C. Liu, and S. Chen, "Smart contract-based campus demonstration of decentralized transactive energy auctions," in *2017 IEEE Power Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, April 2017, pp. 1–5.

[53] PowerLedger, "Power ledger whitepaper," 2018.

[54] M. Faizan, "Ethereum based energy trading platform," Master's thesis, University of Freiburg, 2018.

[55] EWF. (2018) Energy web foundation. [Online]. Available: https://energyweb.org/

[56] ——. Ewf wiki. [Online]. Available: energyweb.atlassian.net/wiki/

[57] ——. Ewf stats. [Online]. Available: http://netstats.energyweb.org/

[58] I. Allision. (2016, Mar.) Microsoft adds ethereum language solidity to visual studio. [Online]. Available: https://www.ibtimes.co.uk/microsoft-adds-ethereum-language-solidity-visual-studio-1552171

[59] Ethereum-Foundation. Solidity v0.4.24. [Online]. Available: https://solidity.readthedocs.io/en/v0.4.24/

[60] B. Rivilin. (2016, Nov.) Geth, viper, and wafr: New ethereum developments. [Online]. Available: https://www.ethnews.com/geth-viper-and-wafr-new-ethereum-developments

[61] Ethereum-Foundation. (2018) Remix, ethereum-ide. [Online]. Available: https://remix.readthedocs.io/en/latest/

[62] web3.py. [Online]. Available: https://web3py.readthedocs.io/en/stable/

[63] web3.js v 1.0. [Online]. Available: https://web3js.readthedocs.io/en/1.0/

[64] Truffle. Ganache. [Online]. Available: https://truffleframework.com/docs/ganache/overview

[65] Ethereum-Foundation. Go ethereum. [Online]. Available: https://geth.ethereum.org/

[66] F. Lange. go-ethereum. [Online]. Available: https://github.com/ethereum/go-ethereum/wiki/geth

[67] B. Arvanaghi. (2018, Jan.) Explaining the genesis block in ethereum. [Online]. Available: https://arvanaghi.com/blog/explaining-the-genesis-block-in-ethereum/

[68] Karalabe. (2017, Mar.) Clique poa protocol & rinkeby poa testnet. [Online]. Available: https://github.com/ethereum/EIPs/issues/225

[69] Bundesnetzagentur. SMARD - Strommarktdaten. [Online]. Available: https://www.smard.de/home

[70] T. Tjaden, J. Bergner, J. Weniger, and V. Quaschning, "Representative electrical load profiles of residential buildings in germany with a temporal resolution of one second," 2015.

[71] L. M. Holm. (2018) Zusammensetzung Strompreis 2017. [Online]. Available: https://1-stromvergleich.com/strom-report/strompreis#strompreis-2017