Open Blockchain-based Local Energy Market Simulation Platform

Master Thesis

Author: Niklas Reinhold (Student ID: 5377277)
Supervisor: Univ.-Prof. Dr. Wolfgang Ketter
Co-Supervisor: Philipp Kienscherf

Department of Information Systems for Sustainable Society
Faculty of Management, Economics and Social Sciences
University of Cologne

August 31, 2019

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne die Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden. Ich versichere, dass die eingereichte elektronische Fassung der eingereichten Druckfassung vollständig entspricht.

Die Strafbarkeit einer falschen eidesstattlichen Versicherung ist mir bekannt, namentlich die Strafandrohung gemäß § 156 StGB bis zu drei Jahren Freiheitsstrafe oder Geldstrafe bei vorsätzlicher Begehung der Tat bzw. gemäß § 161 Abs. 1 StGB bis zu einem Jahr Freiheitsstrafe oder Geldstrafe bei fahrlässiger Begehung.

Niklas Reinhold

Köln, den 02.09.2019

Abstract

Due to the constantly increasing generation from distributed renewable energy resources and the difficulties of integrating these energy resources into the existing electric grid, new market approaches are needed. Peer-to-peer energy trading in local energy markets provides a possible solution to the stated issues, but it requires advanced communication und data exchanges between the different parties, which makes central management and operation more and more challenging. As local energy markets need local distributed control and management techniques, a new and innovative information communication technology is required, to which the emerging distributed ledger technology provides a possible solution. Therefore, an open blockchain-based local energy market simulation platform is developed in this research. The implementation based on a market-based optimization algorithm, introduced by (Guo et al., 2007), that solves a distributed system optimization problem by self-interested agents iteratively trading bundled resources in a double auction market. The underlying information communication technology of the market-based optimization algorithm is provided by the emerging blockchain technology. The conducted research provides a decentralized implementation of the proposed market-based optimization algorithm, in which the transparency is enhanced and the required level of trust is reduced due to the application of the blockchain technology. Furthermore, it provides a detailed explanation of the implementation and deals with the special features of the blockchain functionality, whereby technical barriers in using the developed simulation are removed and researchers are incentivized to design and test their artifacts by using the developed open blockchain-based local energy market simulation platform.

Contents

List of Figures

List of Tables

Listings

List of Abbreviations

ABI   Contract Application Binary Interface

BDP   Bundle Determination Problem

BLEMS Blockchain-based Local Energy Market Simulation

BTM   Bundle Trading Market Framework

CB   Competitive Benchmarking

DLT   Distributed Ledger Technology

EMTS   Energy Management Trading System

ICT   Information Communication Technology

LEM   Local Energy Markets

LP   Linear Program

MMP   Market Matching Problem

PV   Photovoltaic Systems

PoW   Proof of Work

P2P   Peer-to-Peer

RES   Renewabe Energy Sources

1   Introduction

Global warming is one of the most crucial challenges of our time. A shift to sustainable energy sources is needed, which necessitates the cooperation of several disciplines. However, the integration of *renewable energy sources (RES)* into the existing grid is a complex issue and requires new market approaches. This thesis follows the idea of *local energy markets (LEM)* and combines the themes of distributed resource optimization and the emerging *distributed ledger technology (DLT)* to develop a platform for simulating blockchain-based *local energy markets (LEM)*.

1.1   Research Motivation

The generation from distributed RES is constantly increasing (Mengelkamp, Gärttner, et al., 2018). In contrast to power plants which run by non-renewable fossil fuels, distributed RES produce energy in a decentralized and volatile way, which is hard to predict. These characteristics of the distributed RES challenge the current energy system (Ampatzis et al., 2014). The existing electric grid is built for centralized generation by large power plants and the design of the current wholesale markets is not able to react in real-time to a significant amount of distributed RES (Mengelkamp, Gärttner, et al., 2018). Moreover, this way of energy generation is economically not ideal because of energy losses due to long physical distances between generation and consumption parties. Therefore, new market approaches are needed, to successfully integrate the increasing amount of distributed RES (Mengelkamp, Notheisen, et al., 2018). A possible solution to the technical and market problems is *peer-to-peer (P2P)* energy trading in LEM (Long et al., 2017). LEM, also called microgrid energy markets, consist of small scale prosumers, consumers and a market platform that enables the trading of locally generated energy between the parties of a community. Due to the trading of locally generated energy within the related communities, LEM support sustainability and the efficient use of distributed RES. Likewise, the need for expensive and inefficient transportation of energy through long physical distances can be reduced. The concept of LEM strengthens the self-sufficiency of communities and enables possible energy cost reductions. Moreover, profits remain within the communities by which reinvestments in additional RES are promoted (Mengelkamp, Gärttner, et al., 2018).

However, P2P energy trading in LEM requires advanced communication and data exchanges between the different parties, which makes central management and operation more and more challenging. The implementation of LEM need local distributed control and management techniques (Andoni et al., 2019). Therefore,

a new and innovative *information communication technology (ICT)* is required. The emerging DLT provides a possible solution. It is designed to enable distributed transactions without a central trusted entity. A blockchain allows the automated execution of smart contracts depending on vesting conditions, which suits the need of LEM for decentralized and autonomous market mechanisms. This offers new approaches and market designs. Accordingly, DLT can help to address the challenges faced by decentralized energy systems. However, DLTs are not a mature technology yet and there are several barriers in using them, especially for the researcher who do not have a technical background.

Due to the plurality of involved parties and the interdisciplinary requirements for the implementation of new energy market approaches, the accessibility is of major importance. Ketter et al. (2015) introduce the approach of *Competitive Benchmarking (CB)*. This approach describes a research method that faces a real-world wicked problem that is beyond the capacity of a single discipline. It is realized by developing a shared paradigm that is represented in a concrete open simulation platform. In detail, it consists of the three principal elements *CB Alignment*, *CB Platform* and *CB Process*. The CB Alignment refers to the constant synchronization process between the shared paradigm and the wicked problem. The CB Platform represents the medium in which the shared paradigm is technically illustrated. and the CB Platform provides the infrastructure for the third element CB Process. It describes the iterative development of new theories and design artifacts through independent researchers, who influence each other and improve their work in direct sight of each other. The presence of such an open simulation platform depicting the shared paradigm of LEM, would ensure the accessibility and could help to gain new valuable outcomes in the research field of LEM or new energy market designs in general.

Further, the welfare optimization of all participants in LEM can be represented by the central problem of the *Bundle Trading Market Framework (BTM)* developed by Guo et al. (2007). It constitutes a market-based optimization algorithm that solves a distributed system optimization problem by self-interested agents iteratively trading bundled resources in a double auction market run by a dealer. The dealer maximizes the welfare through allowing agents to trade their preferred bundles of energy. Hence, the stated BTM implemented on the basis of a blockchain as underlying ICT can represent the concept of LEM.

This research brings all the introduced approaches together and develops an open *blockchain-based LEM Simulation (BLEMS)*, which enables the research approach based on the three stated elements of CB. The BLEMS is realized through the introduced optimization algorithm with a blockchain as the underlying ICT. That means, the smart contract takes the role of the market dealer and the self-

interested agents represent the individual participants. This paper focuses on the implementation and software design of the open BLEMS.

2   Literature Review

This section outlines the different research streams which are combined by this thesis. This includes previous work in the area of blockchain-based energy markets, distributed resource optimization and competitive benchmarking.

2.1   Blockchain-based Energy Markets

This subsection presents previous work on the topic of blockchain-based energy markets.

Mihaylov et al. (2014) were the first who addressed blockchain technology in energy markets. They presented a new decentralized digital currency for prosumers to trade locally produced renewable energy. In their introduced concept, the generation and consumption of renewable energy are directly transferable into virtual coins. The market value of the virtual currency is determined centrally by the distributed system operator. Al Kawasmi et al. (2015) introduced a blockchain-based model for a decentralized carbon emission trading infrastructure. Their model is based on the bitcoin protocol and focuses on privacy and system security goals. Aitzhan and Svetinovic (2018) addressed the issue of transaction security in decentralized smart grid energy trading and implemented a proof-of-concept for a blockchain-based energy trading system including anonymous encrypted messaging streams. They have shown that blockchains enable the implementation of decentralized energy trading and that the degree of privacy and security is higher than in traditional centralized trading platforms. Sikorski et al. (2017) presented a proof-of-concept where a blockchain enables machine-to-machine (M2M) interactions depicting an M2M energy market. They pointed out that the blockchain technology has significant potential to support and enhance the fourth industrial revolution. Mengelkamp, Gärttner, et al. (2018) presented the concept of a blockchain-based local energy market without the need of a thrusted third entity. They deduced seven market components as a framework for building efficient microgrid energy markets. Consequently, the Brooklyn Microgrid project is introduced and evaluated according to the market components. The Brooklyn Microgrid has also shown that blockchains are a suitable technology to implement decentralized microgrid energy markets, though current regulation does not allow running such LEM in most of the countries. Mengelkamp, Notheisen, et al. (2018) presented a initial proof-of-concept of a simple blockchain-

based concept, market design and simulation of a local energy market consisting of a hundred households. They concluded that the real-life realization and technological limitations of such blockchain-based market approaches need to be investigated by further research. In addition, it is mentioned that regulatory changes will play an important role in the future of blockchain-based LEM.

2.2   Distributed Resource Optimization

The interpretation of dual variables in optimization problems as shadow prices provides a natural understanding of mathematical programs for resource allocation as models of economic activity. The developed decomposition techniques by Dantzig and Wolfe (1961) deepen this understanding, as they allow different actors to emerge from the program's structure. However, when accounting for (adverse) incentives, it is clear that traditional decomposition approaches can be understood rather as a computational than an organizational technique.

Bertsekas (1992) provides an explicit interpretation of network flow problems from an economic viewpoint and Clearwater (1996) gives an extensive overview of control problems that can be approached by economically-inspired techniques.

This stream of research is continued by Fan et al. (2003). They outlined a new approach for the development of an information system that can be used for the problem of a supply chain. The concept demonstrated a decentralized decision-making process that is realized through the design of a market-based coordination system which incentivizes the participants to act in a way that is beneficial to the overall systems. Guo et al. (2007) refined this concept and developed the BTM, a market-based decomposition method for decomposable linear systems, which can be easily implemented to support real-time optimization of distributed systems. They proved that the system optimality can be achieved under a dynamic market-trading algorithm in a finite number of trades. The outlined algorithm can be operated in synchronous as well in asynchronous environments. Guo et al. (2012) extended their stated concept to a dynamic, asynchronous internet market environment. They examined how various market design factors like dealer inventory policies, market communication patterns, and agent learning strategies affect computational market efficiency and implementation. They proved finite convergence to an optimal solution under all these different schemes.

2.3   Competitive Benchmarking

This subsection presents the stream of research of the applied research design.

March and Smith (1995) described a two-dimensional framework for research in IS. The two dimensions can be distinguished into behavioral-science and design-

science. The behavioral-science paradigm is based on explaining and predicting human or organizational behavior. The design-science paradigm is based on broad types of outputs produced by design research. It strives to extend the boundaries of human and organizational capabilities through the creation of new artifacts.

The research framework presented by Hevner et al. (2008) combines both stated IT research paradigms and illustrates the interaction between these two. They presented that technology and behavior are inseparable in an information system and argued that considering the complementary research cycle between design-science and behavioral-science is crucial to address fundamental problems faced in the productive application of information technology.

Ketter et al. (2015) introduced the IS research approach CB, which includes various basic principles of the design science approach of Hevner et al. (2008). This IS research concept is designed for so-called wicked problems and addresses the problems and needs of interdisciplinary research communities. CB focuses on the interconnection of problems at the same and different levels to imitate the real world. In repeated competitions, in which individual teams compete against each other, the developed environment and models can be tested and evaluated. This results in a diversity of outcoming designs, which are difficult to achieve in traditional design science frameworks.

3   Research Design

This research is inspired by the stated CB research approach and focuses on the development of the second CB element, an open simulation platform representing the shared paradigm of LEM. The proposed design is presented in Figure 1. In contrast to the initial CB research design, the third element, CB Process, differs. The platform enables research groups to design and test their artifacts in the field of the shared paradigm. The designed artifacts in the form of autonomous software agents do not compete against each other in competitions. Rather, research groups can design and develop the behavior of all agents and the autonomous market mechanism. Hence, it is also possible to analyze and evaluate the designed artifacts and use them as a benchmark to compare different designs.
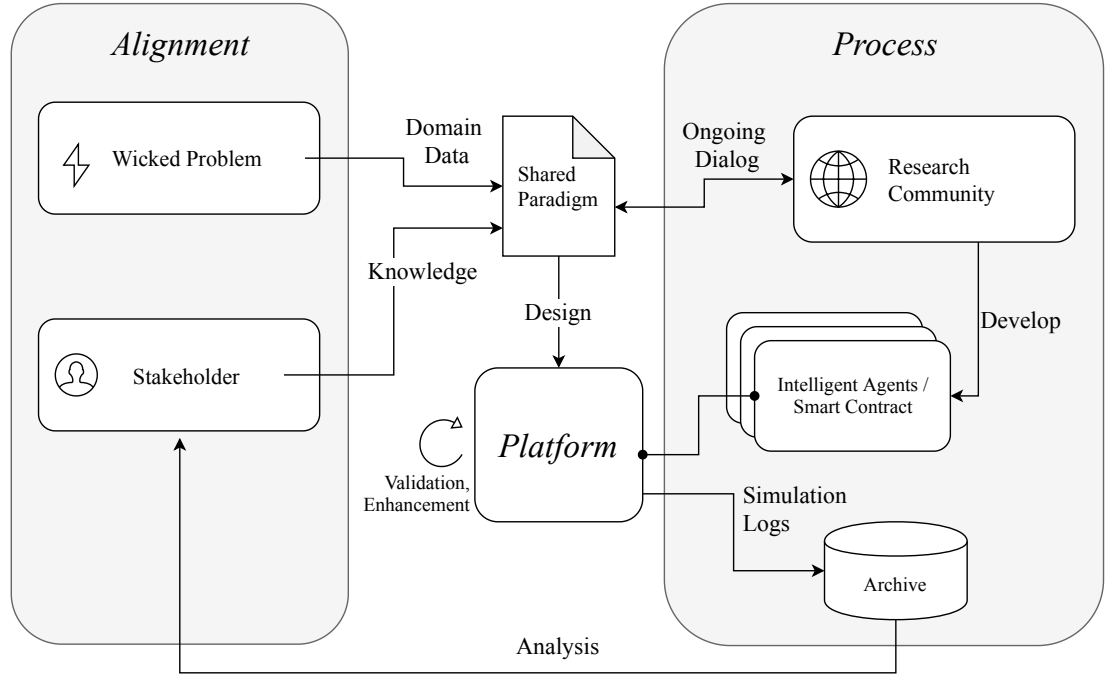
Figure 1: Research Design following Ketter et al. (2015)

According to the IS Design Science principles introduced by Hevner et al. (2008), the presented research design produces a viable artifact in the form of an open simulation platform, which represents a relevant real-world wicked problem. Due to the outcoming data produced by the simulation platform, it is possible to develop methods to evaluate the utility, quality, and efficacy of the artifact. The research provides a verifiable contribution through the artifact, the open simulation platform, itself. The platform enables the investigation of possible solutions to unsolved problems and further, it removes the technical barriers for the use of the complex DLT. The implementation of the platform is based on an appropriate selection of techniques. As stated in 2.1, blockchains are a suitable technology

to implement decentralized microgrid energy markets. The in 2.2 introduced BTM evidentially achieves system optimality under a dynamic market-trading algorithm. This research relies upon the application of rigorous methods and complies with the requirement of the fifth guideline by Hevner et al. (2008). The platform enables the iterative search for an optimal design through the comparison of different produced solutions and is valuable to technology-oriented as well as management-oriented audiences. As a result, this research design also fulfills the seven research guidelines of the research framework introduced by Hevner et al. (2008).

4   Expected Contribution

We expect two different contributions from this research.

First: the full decentralization of the introduced BTM. The bundle-trading market grants access to any given trade to the market dealer. This, again, necessitates the trust of the agents that it will use those resources according to the overarching organizational goal. Due to the implementation of the market dealer by a smart contract, the technical implementation is publicly accessible. Moreover, all transactions of the market dealer to allocate the resources are transparent. Therefore, the behavior of the market dealer is comprehensible for every participant. Furthermore, it provides a high degree of security due to cryptographic encryption methods which are essential parts of the blockchain.

Second: the open *Blockchain-based LEM Simulation (BLEMS)* itself. From the scientific perspective, the platform is relevant due to the removal of existing technical barriers for practitioners in using complex distributed ledgers technologies. Therefore the platform facilitates researchers without a deep technical background to use those DLTs and incentivizes them to design and test their artifacts by using this platform. Furthermore, from a business perspective, this research is relevant to policymakers and energy suppliers. The platform allows stakeholders to get a better understanding of the dynamics of decentralized LEM and enables the testing and evaluating of policy options and implications.

5   Local Energy Markets

This section explains the general concept of LEM and presents possbile market components for the efficient building of LEM.

Due to the centralized generation by large power plants and the current design of the wholesale markets, the existing grid is not suitably designed to react in real-time to a significant growing amount of distributed RES (Ampatzis et al., 2014). To successfully integrate and use these RES, new approaches are necessary (Mengelkamp, Notheisen, et al., 2018). P2P energy trading in LEM provides a possible solution the technical and conceptual problems (Long et al., 2017). In the traditional centralized energy system, large power plants that operate according to a centralized coordination mechanism supply a large amount of customers with energy, who are located within a wide area (for example a country or a state). In contrast, decentralized energy systems consist of small-scale energy generators that are only used by a small number of people and located close to the energy consumption point (Mengelkamp, Gärttner, et al., 2018). Those LEM provide a market platform, market mechanism, and market access for small-scale prosumer and consumer, to trade locally generated energy within their community. A community presents a group of geographically and socially close energy agents. All participants buy or sell energy directly with each other by using the provided market platform without intermediation by conventional energy suppliers (Zhang et al., 2017). If prosumers have a surplus in electricity, they have the opportunity to curtail it, store it in an energy storage device or export it back to the main power grid (Zhang et al., 2017). In the traditional centralized energy system, trading of energy is mainly unidirectional. As stated above, electricity is usually transmitted from large-scale power plants to consumers over long distances, while the cash flow goes the opposite way. On the contrary, the P2P energy trading in LEM encourages multidirectional trading within a local geographical community (Zhang et al., 2017), as illustrated in Figure 2.

Due to this, LEM promote the consumption of energy close to its generation and, therefore, support sustainability and the efficient use of local resources (Mengelkamp, Gärttner, et al., 2018). LEM enable (near) real-time pricing and facilitate a local balance of supply and demand (Mengelkamp, Notheisen, et al., 2018). As stated by Mengelkamp, Gärttner, et al. (2018), the participants of LEM do not necessarily have to be physically connected. Virtual LEM describe the aggregated control of multiple energy producers, prosumers and consumers in a virtual community. Referring to Jimeno et al. (2011), non-virtual LEM have a specific characteristic that distinguishes it from other aggregation of DER systems. They have the opportunity to operate either connected to the main grid or
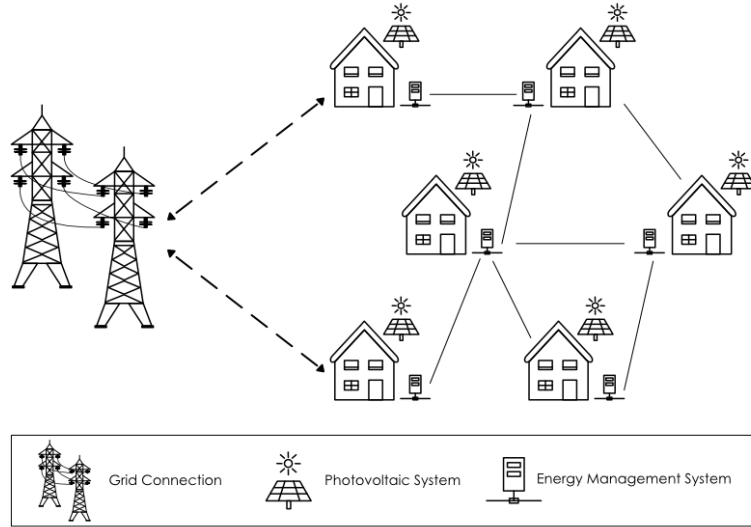
Figure 2: Example of Microgrid setup (Mengelkamp, Gärttner, et al., 2018)

islanded from it. In other words, this allows non-virtual LEM to run disconnected from the main grid, in case it fails or the power quality is not satisfactory. P2P energy trading in LEM requires advanced communication and data exchanges between the different parties, which makes central management and operation more and more challenging. The implementation of LEM needs local distributed control and management techniques (Andoni et al., 2019). Zhang et al. (2017) state that P2P energy trading is often enabled by ICT-based online services. Moreover, Mengelkamp, Gärttner, et al. (2018) explain that the new and innovative blockchain technology as an emerging ICT, offers new opportunities for decentralized market designs. It is designed to enable distributed transactions without a central trusted entity. Accordingly, blockchain can help to address the challenges faced by decentralized energy systems.

5.1   Components of Local Energy Markets

Mengelkamp, Gärttner, et al. (2018) developed seven components for efficient operation of blockchain-based LEM. This subsection names and briefly illustrates each component. Further, the compliance of BLEMS and the seven components will be examined later on in section 11.1. We note that these elaborated components have no general validity and it could also be possible to efficiently implement LEM without providing these components. However, we use these components for an initial evaluation of the platform and its capabilities.

Microgrid Setup (C1): In general, an explicit objective, a definition of the market participants and the form of the traded energy must be well de-

fined. LEM can have different, often contradictory objectives. Especially in the design of the market mechanism, the implementation of the objective plays an important role. Next, a significant number of market participants is needed, who trade energy among each other. Moreover, a part of the market participants needs to be able to produce energy. Finally, the form of the traded energy must be described, for example, electricity, heat or a combination of them. Additionally, the way of energy transportation must be specified. That means, is the the traditional energy grid or a physical microgrid used for energy transportation.

Grid Connection (C2:) The connection points to the superordinate main grid must be well defined. These points measure the energy flows towards the main grid and evaluate the performance of the LEM and can help to balance the energy generation and demand within it. Besides, you have to distinguish between a physical microgrid and a virtual microgrid. A physical microgrid brings along a power distribution grid and is able to decouple from the main grid, whereas a virtual microgrid simply connects all participants over an information system (C3). Consequently, a virtual microgrid does not have the opportunity to physically decouple from the main grid. Nevertheless, to operate in island-mode extensively, a physical microgrid need a large amount of own energy generation capacity and flexibility to ensure supply security and robustness.

Information System (C3): All participants must be connected and a market platform that monitors all operations must be provided. Therefore, an information system is needed, which should enable equal access for every market participant to avoid discrimination. With reference to Mengelkamp, Gärttner, et al. (2018), these requirements can be implemented by blockchain technology based on smart contracts.

Market Mechanism (C4): A market mechanism implemented through the information system is necessary. This market mechanism determine the allocation of the market and the payment rules. Further, a clear bidding format should be defined. It follows that the main objective of the mechanism is to provide an efficient energy allocation by matching the buy and sell orders of the participants appropriately. Finally, this should happen in near real-time granularity.

Pricing Mechanism (C5): The market mechanism (C4) includes the pricing mechanism and supports the efficient allocation of energy supply and demand. The traditional energy price is composed of large parts of taxes and

surcharges. On the contrary, in LEM different fees come to bear, for example in case of a physical microgrid. Hence, RES typically have almost zero marginal cost, prosumer can price their energy above all appropriate taxes and fees to make profit. Thus, the energy price should be linked to the availability of energy. In other words, a surplus of energy should lower the LEM energy price while a lack of energy increases the market price. From an economic point of view, LEM are beneficial to their participants as long as the average energy price is lower than the external grid price.

Energy Management Trading System (C6): The task and goal of the *energy management trading system (EMTS)* is to automatically ensure energy supply for a respective market participant. Therefore, the EMTS needs access to the energy-related data of the participant, like the real-time demand and supply. The EMTS uses this data to forecasts consumption and generation and develops a bidding strategy accordingly. Moreover, the EMTS trades the predicted amounts on the provided market platform and aims to maximize the revenue and minimize the energy costs. For this reason, the EMTS needs to have access to the market participant's blockchain account to be capable to automatically perform energy transactions.

Regulation (C7): It needs to be determined how LEM fit into the current energy policy and which market design is allowed, how taxes and fees are distributed and billed. Likewise, it needs to be determined in which way the local market is integrated into the traditional energy market and energy supply system. All these emerging issues are specified by the legislative regulation.

6 Blockchain

We stated that the implementation of LEM needs local distributed control and management techniques and emphasize that the blockchain technology as an emerging ICT offers new opportunities for decentralized market designs and helps to address the challenges faced by decentralized energy systems. Therefore, this research applies a blockchain in the BLEMS as the underlying ICT, which is why this chapter gives an overview of the general purpose, the contained components and the fundamental functionality of a blockchain.

6.1 General Purpose

In general, a blockchain is a digital data structure that can be understood as a shared and distributed database, containing a continuously expanding and chronological log of transactions (Andoni et al., 2019). Various data types like digital transactions, data records and executables can be stored in this digital data structure. The data transmission in a blockchain is comparable with copying data from one computer to another. However, the resulting challenge is that the system needs to ensure that the data is copied just once (Andoni et al., 2019). For example, in the domain of cryptocurrencies, this is equal to sending a coin from one wallet to another. The system needs to validate that this coin is spent just once and there is no double-spending. A conventional solution for this problem is a third intermediary. To give a tangible example, the third intermediary can be described by a traditional bank, which stores, protects and continuously updates the valid state of the ledger (Andoni et al., 2019). In some cases, central management is not practicable or reasonable due to possible intermediary costs or a high degree of trust of the users into the intermediary who operates the system. Central management has a significant disadvantage because of a single point of failure. Hence, the centralized system is fragile to technical problems as well as to external malicious attacks (Andoni et al., 2019). Consequently, the main reason for blockchain technologies is the removal of such third trusted intermediaries through a distributed network of various users, who cooperating to verify transactions and protect the validity of the ledger.

6.2 Components

This subsection covers the architectural design of a blockchain and presents the contained components in detail. Due to the plurality of the blockchain technologies, each of the technology slightly differs in design and components. The following explanations are oriented towards the Ethereum, as it used in the im-

plementation of the BLEMS.

6.2.1   World State

Referring to Wood (2014), Ethereum can be seen as a transaction-based state machine. As you can see in Figure 3, the state machine starts with a so-called *genesis state*. This is analogous to a blank sheet. In this state, no transactions have happened on the network. Next, transactions are executed incrementally and the state of the Ethereum world changes into new states and morph it into some final state. Consequently, the final state is accepted as the canonical version of the world of Ethereum and represents at any time the current state.
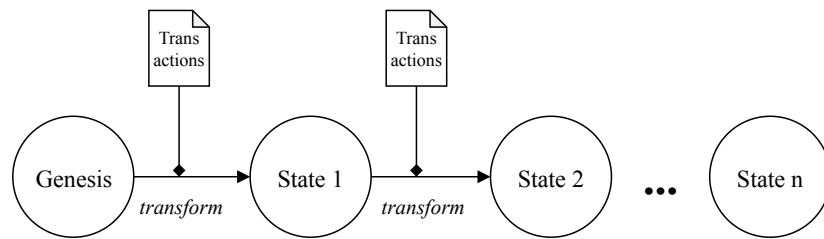
Figure 3: World State: Transition of States

The world state arises out of a mapping of a key-value pair for every account which exists on the Ethereum network (Wood, 2014). The key constitutes the address of an Ethereum account and the value presents the account's state, which contains detailed information. However, the world state is not stored on the blockchain itself. This mapping is stored and maintained in a modified data structure called a *Patricia Merkle tree*. This tree is stored off-chain in a simple database backend (i.e. on a computer running an Ethereum client), also known as the *state database* in the Ethereum world (Wood, 2014). To get a better understanding of the operating principles of the blockchain, it is necessary to get an idea of how a Patricia Merkle tree works. A Patricia Merkle tree is a type of binary tree, which consists of a set of nodes. It has a large amount of leaf nodes, containing the underlying data. Further, a set of intermediate nodes, where each node is the hash of its two children, and finally, one single root node, representing the top of the tree that is also built out of its two child nodes (Buterin et al., 2013) (Wood, 2014). As mentioned before, the leaf nodes contain the stored data by splitting these data into *chunks*. Afterwards, these *chunks* are split into *buckets*. Then, each bucket gets hashed and the same process repeats, traversing upwards the tree until the total number of hashes remaining becomes only one and the root node is reached (*Merkling in Ethereum*, 2015).

Therefore, any change to the underlying data, stored in a *leaf node*, causes a change of the hash of the node. Each parent's node hash depends on the data of
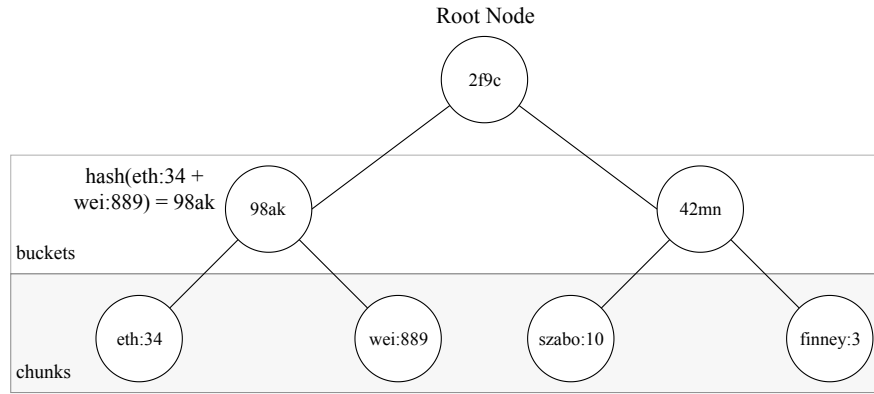
Figure 4: Example of a Patricia Merkle tree

its children. Due to this, any change to the data of a child node causes the parent hash to change. This procedure repeats traversing upwards until the root node. Hence, any change to the data at the leaf nodes affects the root hash (*Modified Merkle Patricia Trie Specification*, 2015).
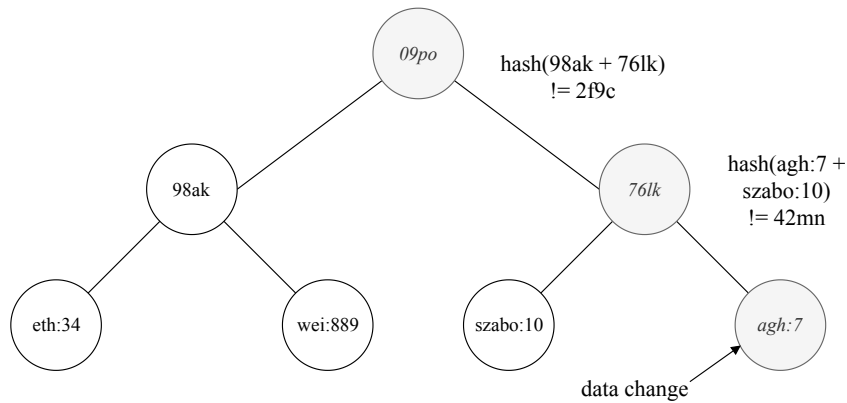
Figure 5: Example of a data change in a leaf node

Because of this characteristic, it is not necessary to compare the data of the entire tree. It is sufficient to compare the single root hash to ensure that all the data are the same. This property is very important because it makes it possible to store only the hash of the root node to represent a state of the Ethereum world.

6.2.2   Block

As described in section 6.2.1, whenever transactions are executed, the state of the Ethereum world changes into a new state. Every state and the belonging transactions, which transform the prior state into the new state, is cumulated in a so-called block. That means the history of the Ethereum world is a linkage of states which are represented by blocks. That is where the name blockchain comes from. A blockchain is a sequence of blocks, which holds a complete list

of transaction records (Zheng et al., 2017). Moreover, a block is a collection of different relevant information and consists of the *block header* and the *block body*, which contains the list of transactions. Following Wood (2014), the subsequent pieces of information are contained in the block header:

Parent Hash: Hash of the parents' block's header. Therefore, every block points to his ancestor. Due to this contained attribute, a chain arises out of the single blocks.

Beneficiary: Miners address to which all block rewards from the successful mining of a block are transferred.

State Root: Hash of the root node of the state tree after a block and its transactions are finalized. As mentioned in section 6.2.1, the state tree is the unambiguous global state in the Ethereum world. It is used as a secure unique identifier for the state and the state root node is cryptographically dependent on all internal state tree data.

Transactions Root: Hash of the root node of the transaction tree. This tree contains all transactions in the block body. In contrast to the state tree, there is a separate transaction tree for every block.

Receipts Root: Every time a transaction is executed, Ethereum generates a transaction receipt that contains information about the transaction execution. This field is the hash of the root node of the transactions receipt tree and as the transaction tree, there is a separate receipt tree for every block.

Difficulty: Measure of how hard it was to mine this block – a quantity calculated from the previous block's difficulty and its timestamp

Number: Quantity equal to the number of blocks that precede the current block in the blockchain.

Gas Limit: Quantity equal to the current maximum gas expenditure per block. Each transaction consumes gas. The gas limit specifies the maximum gas that can be used by the transactions included in the block. It is a way to limit the number of transactions in a block.

Gas Used: Quantity equal to the total gas used in transactions in this block.

Timestamp: Record of Unix's time at this block's inception.

Nonce: Hash that verifies a sufficient amount of computation has been done on this block. Further, it is a number added to a hashed block that, when

rehashed, meets the difficulty level restrictions. The nonce is the number that blockchain miners are solving for.

We introduced Ethereum as a transaction-based state machine, transforming one state into another through the execution of transactions. Further, we explained how these individual states are stored and emphasizes the meaning of this storage. We pointed out that an Ethereum state is represented by blocks and described all relevant information contained in it. However, we didn't introduce how many transactions be part of a block and who build and validate a block? To answer this, we introduce the transaction object and its life cycle in-depth.

6.2.3   Transaction

A transaction is a basic method for Ethereum accounts to interact with each other. In the Ethereum world exists two different types of transactions. Those, which result in a so called *message call*, which can be seen as a traditional transaction, and those which result in a *contract creation* (Wood, 2014). Both different types of transaction share the following common attributes (Wood, 2014):

Nonce: This attribute is discontiguous of the block attribute. In contrast to the block attribute, the nonce of a transaction keeps track of the total number of transactions that an account has executed.

Gas Price: As mentioned in section 6.2.2, each transaction consume gas. Gas can be seen as fees. This attribute presents the price per unit of gas for a transaction [1].

Gas Limit: This attribute is similar to the block attribute of the same name. In this case, it is a quantity equal to the current maximum gas expenditure per transaction.

To: In case of a message call, this attribute contains the address of the recipient. Otherwise, in case of a contract creation this value remains empty.

Value: In case of a message call, this attribute contains the amount of Wei, which will be transferred to the recipient. In case of a contract creation, this value contains the amount of Wei as an initial endowment of the contract. Wei is a subdivision of the cryptographic currency of Ethereum and 1 Wei is defined as 10^{-18} Ether.

---

[1]A website to see current gas prices: https://ethgasstation.info/index.php

6.2.4 Transaction Life Cycle

The life cycle of a transaction is outlined. We take a simple message call as an example and pass through the entire flow of how this transaction gets executed and permanently stored on the blockchain. During this process, many basic concepts of a blockchain are introduced.

To begin with, someone wants to send an arbitrary amount of ether to someone else. Consequently, a transaction with the respective attributes, which are stated in section 6.2.3, will be created. As a first step, this transaction will be signed. That means the one who wants to execute the transaction needs to prove the ownership of the account. Otherwise, someone else could execute a transaction on your behalf. The way to prove this is by signing the transaction with the corresponding private key of this account (*Medium, Life Cycle of Transactions*, 2017). Next, the signed transaction is submitted to the related Ethereum node. Then, the node will validate the signed transaction. After successful validation, the node will send the transaction to its peer nodes, who again send it to their peer nodes and so on (*Medium, Life Cycle of Transactions*, 2017). As soon as the transaction is broadcast to the network, the related node will declare a transaction ID, which is the hash of the signed transaction and can be used to track the status of the transaction [2].
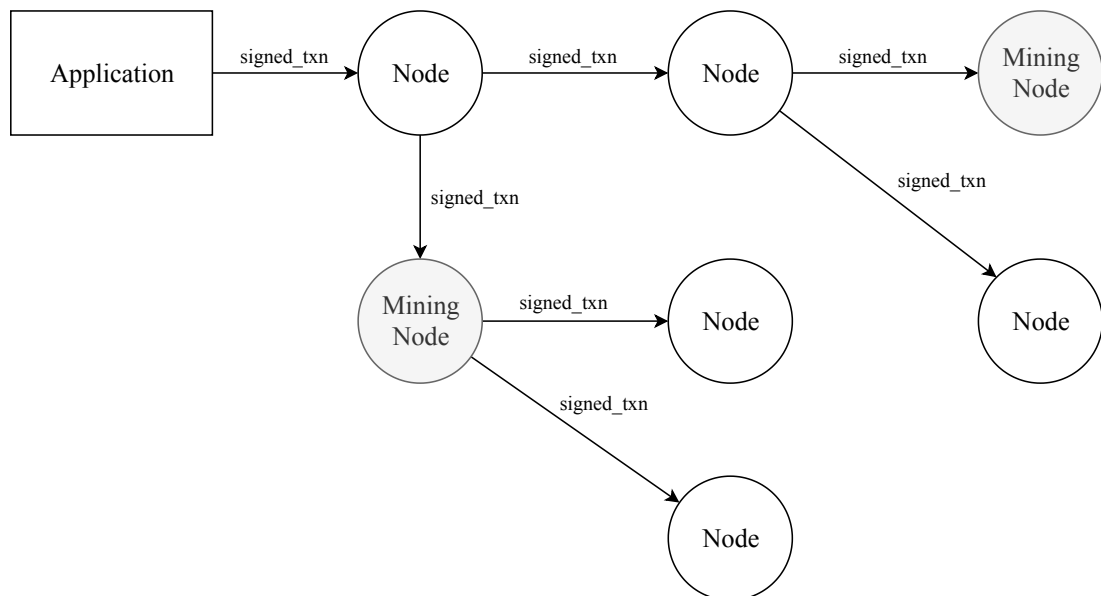

Figure 6: Node network demonstrating a transaction broadcast

---

[2]A Website to track transactions: https://etherscan.io/

Further, Figure 6 describes an Ethereum network. It contains a mix of miner nodes and non-miner nodes. The non-miner nodes can be distinguished into different types of nodes with different behavior, which are explained in detail in section 6.2.6. The miner nodes are the ones who processing transactions into blocks (*Medium, Life Cycle of Transactions*, 2017).

Mining nodes contain and maintain a pool of transactions where they collect incoming transactions. They sort the transactions by gas price. However, there are no specific rules on how the nodes sort the transactions. A common configuration is to sort the transaction by gas price in descending order to optimize for a higher pay (*Medium, Life Cycle of Transactions*, 2017).

Table 1: Transaction Pool of Mining Node

Signed Transaction	Gas Price
hashOfTransaction1	15 Gwei
hashOfTransaction2	13 Gwei
hashOfTransaction3	11 Gwei
hashOfTransaction4	10 Gwei
hashOfTransaction5	7 Gwei

Further, the mining node takes transactions from the pool and processes them into a pending block. A block can only contain a certain number of transactions due to the block attribute *gas limit*, mentioned in section 6.2.2. That means the mining node can only process so many transactions into one block until the gas limit is reached [3]. For example, imagine that the current gas limit of a block is 28 Gwei. Next, we look at the transaction pool of a mining node in table 1. Two blocks are required to process all transactions from the pool. The first two transactions are contained in the first block and the second block contains the remaining three transactions. So far, it is described how a pending block will be created and the transaction limit of a block is explained. In the next section 6.2.5, the process of a pending block to a valid block is outlined in depth. After the validation process of a block is finished successfully, the life cycle of a transaction is terminated. In other words, the transaction is anchored in the blockchain and modified the world state as mentioned in section 6.2.1.

6.2.5   Mining

In this section, the steps of the validation process of a pending block is described. We explain which node builds a new block and how this regenerated block is accepted by the rest of the network. The process responsible for this is called

---

[3]A website to see current attributes: https://ethstats.net/

consensus mechanism. There are many various consensus algorithms, each of them provides different features, advantages, and disadvantages.

To a large extent, the key performance drivers of a blockchain like transaction speed, scalability and security depend on the embedded consensus algorithm (Andoni et al., 2019).

In this section, we expound the *Proof of Work (PoW)* mechanism, which is at the time of this writing the used consensus mechanism in the Ethereum blockchain. PoW enables high scalability in terms of the number of nodes and clients (Vukolić, 2015), whereas it provides a poor transaction speed and high power consumption due to the solving of a cryptographical puzzle, which requires significant computational effort (Andoni et al., 2019).

PoW is a random process that is not predictable and therefore only solvable through a trial and error approach (*Proof of Work*, 2019). All mining nodes compete with each other and the goal is to achieve a hash output that is lower than a given specified target (Andoni et al., 2019). The hash output includes and comprises out of the *Parent Hash*, *Transaction Root* and *Nonce*, which are all part of the block headers data, mentioned in 6.2.2. However, the Parent Hash and Transaction Root are given and immutable, thus the Nonce is the value that all the mining nodes are solving for[4]. Consequently, the mining nodes modify the Nonce until the hash output is lower than the required target (Andoni et al., 2019). The following pseudocode in Listing 1 illustrates the PoW procedure:

```
targetValue = 00005cdf6d384113165841052dfd4638eaf756ac
parentHash = abbecf2d59eacbde676c3f1f0bfcf124f8c68209
transactionRoot = 917c631e5ee59596859910759c8ed76a21252010
nonce = 1
valid = False

while(not valid):
        hashOutput = sha256(parentHash+transactionRoot+nonce)

        if(hashOutput <= targetValue):
                valid = True
        else:
                nonce += 1
```

Listing 1: Pseudocode of PoW Mechanism

---

[4]A interactive blockchain demo: https://anders.com/blockchain/

Finally, when one mining node successfully calculated the Nonce and therefore reaches the target value, it broadcasts the block to all other mining nodes in the network. Now, all other nodes mutually validate the correctness of the hash value by recalculating the hash output and verifying if it is lower than the target value. If the block is validated through all other mining nodes, all nodes will append this new block to their own blockchains (Zheng et al., 2017).

Considering, the network is decentralized and the simultaneous generation of valid blocks is possible when multiple nodes find the suitable Nonce at a nearly same time (Zheng et al., 2016). In this case, branches will be generated as shown in Figure 7
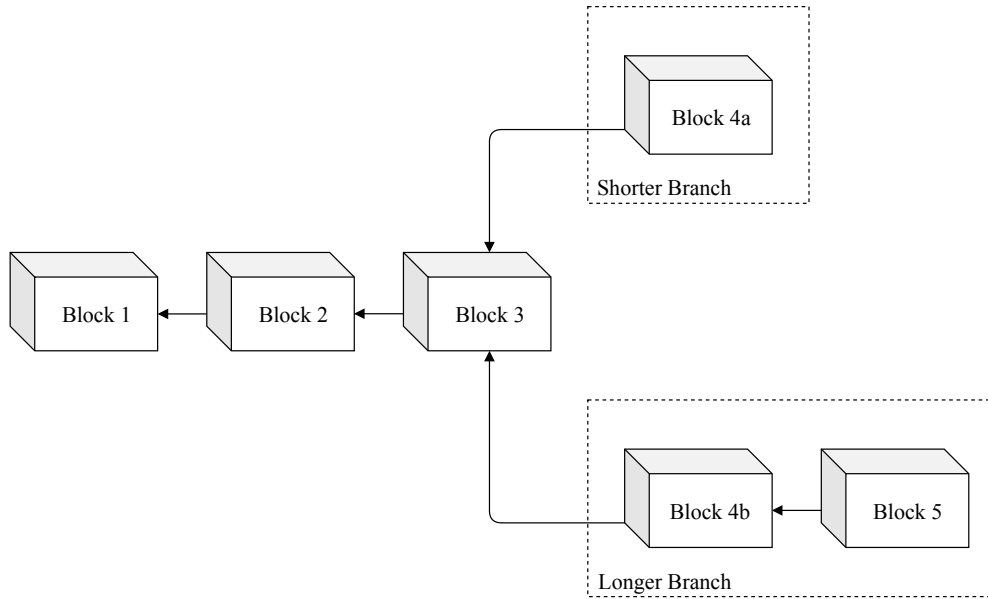
Figure 7: Scenario of Blockchain Branches

Nevertheless, it is unlikely that two competing branches will generate the next block simultaneously again. In the PoW protocol, the longer branch is judged as the authentic one. In Figure 7, two branches arised out of two simultaneous valided blocks Block 4a and Block 4b. After that, all mining nodes accept both branches and start working on the next consecutive block until a longer branch is found (Andoni et al., 2019). As you can see in Figure 7, Block 4b and Block 5 forms the longer chain, therefore all miners on the branch with Block 4a will switch to the longer branch (Zheng et al., 2017).

To sum up, this section joins and completes the previous section 6.2.4. The consensus mechanism PoW is described and outlined in pseudocode, whereby the validation process of newly generated blocks is explained and illustrated.

6.2.6   Ethereum Clients

The expressions Ethereum client and Ethereum node can be used interchangeably. As mentioned earlier in the previous section 6.2.4 and illustrated in the Figure 6, clients can be distinguished into different types. This subsection describes the concept of an Ethereum client in general and gives a brief overview of the different types.

In general, an Ethereum client is a software application that implements the Ethereum specification, which is specified in the Ethereum yellow paper (Wood, 2014). A client communicates over the P2P network with other Ethereum clients. Although these clients can be implemented in different programming languages, they all communicate through the standardized Ethereum protocol, whereby they can operate and interact with the same Ethereum network (Antonopoulos & Wood, 2018).

We present a list of the current main implementations of the Ethereum protocol, since the Ethereum blockchain is not officially implemented in a particular programming language:

- `Parity`, written in Rust

- `Geth`, written in Go

- `cpp-Ethereum`, written in C++

- `pyethereum`, written in Python

- `Harmony`, written in Java

While all clients differ from each other, they share some fundamental features (*Medium, Ethereum-Clients*, 2019).

First, each client can join the P2P Ethereum network and they all synchronizing a local copy of the blockchain. There are a few modes of synchronizing, which comes with miscellaneous advantages and disadvantages, which makes a significant difference in the health and resilience of the decentralized network. However, each client is capable of broadcasting new transactions to the network and creating and managing accounts (*Medium, Ethereum-Clients*, 2019). In general, the clients can be distinguished into two different types, the so-called *full node* and *light node*.

Full Node: A full node downloads all history data P2P from another full node. This requires a significant amount of hardware and bandwidth resources (Antonopoulos & Wood, 2018). Afterwards, it simulates every transaction in the

ledger and execute the whole deployed source code to recalculate the state of each existing block (*Medium, Ethereum-Clients*, 2019). Therefore, the amount of independently operating and geographically dispersed full nodes is a crucial and very important indicator for the health, resilience and censorship resistance of a blockchain (Antonopoulos & Wood, 2018). However, a full node is not necessarily a mining node. It authoritatively validates all transactions, but to participate in the mining competition, an additional software called *ethminer* is needed [5]

Light Node: To begin with, a light node is not a separate piece of software from the clients that run a full node. The difference lies in the light mode for synchronization. In contrast to a full node, a light node only synchronize the block headers and the current state of the chain (*Light-Client-Protocol*, 2019). Therefore, a light node cannot run transactions throughout the history of the whole blockchain. For this reason, it does not contribute to the health and resilience of the network like a full node and traditionally cannot act as a mining node (*Medium, Ethereum-Clients*, 2019). In addition, a light node has some further limitations. For instance, they are not capable to monitor pending transactions from the network and the hash of a transaction is not sufficient to locate the transaction. Moreover, to perform certrain types of operations, a light node relies on requests to full nodes, whereby they can be far slower to query the chain (*Light-Client-Protocol*, 2019).

6.2.7   Smart Contract

This section deals with smart contracts in the Ethereum world. It presents a definition of smart contracts and explains its characteristics.

The term smart contract is a bit misleading, since the Ethereum smart contracts are neither smart nor legal contracts. An Ethereum smart contract is an immutable computer program that runs deterministically in the context of an Ethereum Virtual Machine as part of the network protocol Antonopoulos and Wood (2018). Immutability means that once the contract is deployed into the blockchain, it is not possible to make any changes in the source code. The only way to modify the contract is to deploy a whole new instance. Deterministic means that the result of the execution is the same for all who run it, depending on the context of the transaction that initiated its execution and the state of the blockchain at the moment of execution Antonopoulos and Wood (2018). As already mentioned in section 6.2.3, in the Ethereum world exists two different types of transactions. Those which result in a message call, and those which result in

---

[5]GPU Mining Worker: https://github.com/ethereum-mining/ethminer

contract creation. That means smart contracts are deployed through special contract creation transactions into the blockchain. The special thing about these transactions is that the recipient of the transaction remains empty, whereby the transaction will be sent to a special destination address called *zero address (0x0)*. Each contract is identified and reachable via a common Ethereum address, which can be used in a transaction as the recipient to send funds to the contract or to call one of the contract functions. Regardless, in contrast to an externally owned account, there are no keys associated with an account of a smart contract Antonopoulos and Wood (2018). For this reason, the creator of a smart contract does not have any special rights at the protocol level. Although, it is possible to get those special rights if you explicitly specify them in the source code of the contract. Furthermore, contracts are only active and run if they called by a transaction. A contract can call another contract and so on, but, the first call has always come from an externally owned account. A contract will never run on his own or in the background Antonopoulos and Wood (2018). Additionally, contracts are not executed in parallel. They always execute consecutively, hence the Ethereum blockchain can be considered as a single-threaded machine. It should be noted that transactions are atomic regardless of how many contracts they call. Transactions execute in their entirety and any changes in the global state are only recorded if all executions terminate successfully. If the executions fail, all the changes in state are reverted as if the transaction never ran. Nevertheless, a failed execution of a transaction is recorded as having been attempted and the gas fees spent for the execution are withdrawn from the originating account.

7   Linear Programming

This section gives an introduction into the basics of mathematical linear programming, which forms the basis for the BTM applied in the BLEMS. It presents the purpose and necessity of optimization models, gives an example of how an optimization model is constructed and explains the duality theory, which is essential for determining the market prices in the BTM and for the developed agent trade verification procedure, outlined later on in 10.2.1. Finally, the used algorithm to solve such optimization models is presented.

Optimization models mathematically define the goal of solving a problem in the optimal way. This can be applied in many different areas, for example, it might mean running a business to maximize profit, designing a bridge to minimize weight or selecting a flight plan for an aircraft to minimize time or fuel use (Griva et al., 2009). The use case of solving an optimization problem optimally is so ubiquitous, that optimization models are used in almost every area of application (Griva et al., 2009). Often it is not possible or economically feasible to make decisions without the help of such a model. Due to the excellent improvements in computer hardware and software in the last decades, optimization models became a practical tool in business, science and engineering (Griva et al., 2009). Finally, it is possible to solve optimization problems with a huge set of variables.

Linear optimization, also known as a *linear program (LP)*, refers to the optimization of a linear function with the decision variables, $x_1, ..., x_n$ subject to linear equality or inequality constraints. As presented by Bichler (2017), in canonical form a linear optimization model is given as:

$$
\begin{aligned}
\min \quad & \sum_{j=1}^{n} c_j x_j \\
\text{s.t.} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \quad i = 1, ..., m \\
& x_j \geq 0, \qquad\quad j = 1, ..., n
\end{aligned}
$$

However, a LP can be modeled in different forms. It is also possible to write a model in the matrix-vector notation. To represent the above model in this form, letting $x = (x_1, ..., x_n)^T$, $c = (c_1, ..., c_n)^T$, $b = (b_1, ..., b_m)^T$ and name the matrix of the coefficients a_{ij} by A. As introduced by Griva et al. (2009), the model becomes:

$$
\begin{aligned}
\min \quad & c^T x \\
\text{s.t.} \quad & Ax \geq b \\
& x \geq 0
\end{aligned}
$$

In general, the objective function of a linear model is either minimized or maximized. Additionally, the constraints may include a combination of inequalities and equalities and the variables are unrestricted or restricted in sign (Bichler, 2017).

7.1   Duality Theory

For every LP exists another problem, which is called the *dual* of the original linear problem, also known as the *primal* (Bichler, 2017). In the dual, the roles of variables and constraints are reversed. That means, every variable of the primal becomes a constraint in the dual, and every constraint in the primal becomes a variable in the dual (Griva et al., 2009). For instance, if the primal has n variables and m constraints, the dual will have m variables and n constraints. Further, the primal objective coefficients are the coefficients on the right-hand side of the dual, and vice versa. Finally, the transposed coefficient matrix of the primal constitutes the matrix in the dual (Griva et al., 2009). To give an example of a primal and the corresponding dual LP, the representation by Bichler (2017) is considered:

$$
\begin{aligned}
\min \quad & c^T x \\
\text{s.t.} \quad & Ax \geq b \\
& x \geq 0
\end{aligned}
\qquad \text{(primal)}
$$

$$
\begin{aligned}
\max \quad & b^T y \\
\text{s.t.} \quad & A^T y \leq c \\
& y \geq 0
\end{aligned}
\qquad \text{(dual)}
$$

Next, the motivation of the duality theory can be explained by the issue of trying to find bounds on the value of the optimal solution to a LP. If the primal is a minimization problem, the dual represents a lower bound on the value of the optimal solution, otherwise, if the primal is a maximization problem, the dual represents an upper bound (Bichler, 2017). To illustrate the concept and motivation of duality theory more tangibly, an example stated by Bichler (2017) is given. In an application, the variables in the primal linear problem stand for consumer products and the objective coefficients represent the profits linked to the production of those consumer products. In this case, the objective in the primal describes directly the relation between a change in the production of the products and profit. Whereas, the constraints in the primal describe the availability of raw materials. Consequently, an increase in the availability of raw materials needed for the production of the consumer products enables an increase in production, and finally an increase in the overall profit. However, the primal problem does

not outline this relationship reasonable. Hence, one of the benefits of the dual is to properly show the effect of changes in the constraints on the value of the objective. Due to this, the variables in a dual LP often called *shadow prices*, since they describe the hidden costs associated with the constraints.

7.1.1   Weak Duality Theorem

As stated in the section before (7.1), the *dual* problem provides, depending on maximization or minimization problem, lower and upper bounds for the *primal* objective function. Referring to Vanderbei et al. (2015), this result is true in general and is described as the *Weak Duality Theorem*:

Theorem 1. *If x is feasible for the primal and y is feasible for the dual, then:*

$$c^T x \leq b^T y$$

Regarding to Griva et al. (2009), there are some logical corollaries of the Weak Duality Theorem:

Corollary 1. *If the primal is unbounded, then the dual is infeasible. If the dual is unbounded, then the primal is infeasible.*

Corollary 2. *If x is a feasible solution to the primal, y is a feasible solution to the dual, and $c^T x = b^T y$, then x and y are optimal for their respective problems.*

7.1.2   Strong Duality Theorem

Concerning corollary 2, it is possible to verify if the points x and y are optimal solutions, without solving the corresponding LPs. Furthermore, this corollary is also used in the proof of strong duality (Griva et al., 2009). The *Strong Duality Theorem* describes, that for linear programming there is never a gap between the primal and the dual optimal objective values (Vanderbei et al., 2015).

Theorem 2. *If the primal problem has an optimal solution x^*, then the dual also has an optimal solution y^*, such that:*

$$c^T x^* = b^T y^*$$

7.1.3   Simplex-Method

This subsection will give a brief introduction to the *simplex-method.* In the field of linear programming, the simplex-method is the most widely used algorithm due to its efficiency. (Griva et al., 2009). In general, the simplex-method is

an iterative algorithm, used to solve a LP (Griva et al., 2009). To apply the algorithm to a LP, it must be in standard form (Luenberger, 2008). According to Griva et al. (2009), we consider the following LP:

$$
\begin{aligned}
\min \quad & z = -x_1 - 2x_2 \\
\text{s.t.} \quad & -2x_1 + x_2 \leq 2 \\
& -x_1 + 2x_2 \leq 7 \\
& x_1 \leq 3 \\
& x_1, x_2 \geq 0
\end{aligned}
$$

Next, we give an example to illustrate how to convert a LP into standard form. To achieve that, the so-called *slack variables* will be added, which represents the difference between the right-hand side and the left-hand side.

$$
\begin{aligned}
\min \quad & z = -x_1 - 2x_2 \\
\text{s.t.} \quad & -2x_1 + x_2 + x_3 = 2 \\
& -x_1 + 2x_2 + x_4 = 7 \\
& x_1 + x_5 = 3 \\
& x_1, x_2, +x_3, +x_4, +x_5 \geq 0
\end{aligned}
$$

Finally, with the LP in standard form, the idea of the simplex-method is to iteratively proceed from one basic feasible solution to another (Luenberger, 2008). Thereby, it always looks for a feasible solution which is better than the one before. The definition of better, in this case, depends on the nature of the linear problem. The process is performed until a solution is reached, which cannot be improved anymore. In the end, this final solution is then an optimal solution (Vanderbei et al., 2015).

8   The Bundle-Trading-Market Framework

We explained in section 5 that LEM enable the trading of locally generated energy through a market platform, market mechanism and market access for small-scale prosumer and consumer. In addition, we presented in section 5.1 a market mechanism as one of seven core components for the efficient operation of blockchain-based LEM. On account of that, this chapter covers and explains in detail the applied market mechanism in the BLEMS.

As already briefly introduced in the sections 1.1 and 2.2, the applied market mechanism is implemented through the developed market-based optimization algorithm, which is also called the BTM. In the proposed framework by Guo et al. (2012), independent, self-interested agents trading bundled resources in a double-auction market, which run by a dealer. In this case, the dealer replaces the central authority and agents represent distributed entities. Generally, the BTM consists of a master problem that solves the *market matching problem (MMP)*, which is managed by the dealer and subproblems that solve the bundle-determination problem of the agents. The solution process is an iteration between the MMP and the the bundle-determination problems (Guo et al., 2007). Considering, the BTM constitutes a market-based decomposition method for decomposable linear systems, which can be easily implemented to support real-time optimization of distributed systems (Guo et al., 2007). The central problem of the BTM can be interpreted as the welfare optimization of all participants in LEM. The dealer, which runs the double auction market, maximizes the welfare through allowing agents to trade their preferable bundles of energy. Therefore, the BTM is a suitable market mechanism for the concept of LEM.

8.1   Problem Overview

This subsection gives an introductory description to the overall problem. With reference to Guo et al. (2012), we consider a distributed system with k independent agents. We have a central problem and individual problems of the k individual agents, which both can be expressed as the LPs shown the in equations (1) and (2). The BTM presented by Guo et al. (2007) requires a nondegenerate central problem with a bounded solution. In contrast to discrete markets where only integer number of units can be traded, the proposed framework facilitates continuous trade amounts.

Central Problem

$$Z(c) = \min_{x_j \geq 0} \quad \sum_{j=1}^{k} d_j^T x_j$$

$$\text{s.t.} \qquad N_j x_j \leq n_j, \quad j = 1, ..., k \tag{1}$$

$$\sum_{j=1}^{k} C_j x_j \leq c$$

Agent Problem $(j = 1, ..., k)$

$$z_j(c_j) = \min_{x_j \geq 0} \quad d_j^T x_j$$

$$\text{s.t.} \qquad N_j x_j \leq n_j, \tag{2}$$

$$C_j x_j \leq c_j$$

In the table below, the applied variables in the stated equations are described:

Table 2: Applied Variables in BTM Framework

Variable	Description
$d_j \in R^{b_j}$	vector of agents $j's$ costs
$x_j \in R^{b_j}$	decision variables controlled by agent j
$N_j \in R^{a_j \times b_j}$	activity matrix
$C_j \in R^{m \times b_j}$	activity matrix
$n_j \in R^{a_j}$	capacity vector of agent $j's$ independent resources
$c_j \in R^m$	agent $j's$ vector of shared resources

The overall objective of the central problem is to minimize the operating costs of the overall system. The minimization of the central problem has to be implemented in consideration of the operational constraints (first set of constraints) and the total shared resources capacity constraints (second set of constraints) of each individual agent. Nevertheless, it is assumed that the optimal solution to the central problem cannot be directly calculated, due to the lack of access to all relevant information to make the decisions (d_j, n_j, c_j, N_j, C_j, for $j = 1, ..., k$). Guo et al. (2007) stated that, for an allocation of $c_j, j = 1, ..., k$, such that $C_j x_j^* \leq c_j, \sum_{j=1}^{k} C_j$, solving the agents' problems is equivalent to solving the central problem.

Hence, a market-based resource allocation mechanism to coordinate decentralized decision making from agents is used by the BTM. For any initially given $c_j, j = 1, ..., k$, these mechanism enables an indirectly acquisition of an optimal solution to the central problem through a direct, iterative and wealth-improving

bidding process among the self-interested agents.

8.2   Market Environment

Generally, a dealer and k independent agents constitute the whole market economy. Each of the agents has the opportunity to trade the shared resources c_j in a double auction market, which is operated by the dealer. Each agent only has a local perspective and knowledge, that is to say, they don't know anything about the production decisions of the others. The market prices to match the trades will be determined and published by the dealer. Subsequently, the roles of the market participants are described and the market operations like the agent bidding, market matching, and settlement are outlined. Finally, a summary of the market-based algorithm is presented.

8.2.1   Agent Bidding

The initialization takes place, which means that each of the $j = 1, ..., k$ agents receive an initial endowment of the m shared resources c_j , and a cash endowment e_j. The wealth of an agent at any point is defined as $e_j - z_j(c_j)$ (Guo et al., 2007). The agents have the opportunity to buy additional resources or to sell some of their own resources to lower their operating costs. Therefore, each agent knows the current market prices $p \in R^m$ for the shared resources, which are published by the dealer after each auction round. For trading the shared resources in the market, a bundle referring to Guo et al. (2012) is defined:

Bundle   A bundle $w \in R^m$ is an m-dimensional vector of shared resources. Each of the m elements corresponds to an amount of one specific shared resource. A negative sign of an element signifies a sell amount, and contrary, a positive sign signifies a buy amount.

An *improving bundle set* concerning to lower operating costs is defined as the following:

Improving Bundle Set

$$W_j(c_j|p) = \{w : \exists x_j \geq 0 \ni d_j^T x_j + p^T w \leq z_j(c_j),$$
$$N_j x_j \leq n_j, C_j x_j \leq c_j + w\} \tag{3}$$

To remember that $z_j(c_j)$ is the optimal value of the agent $j's$ problem depending on the amount of the shared resources c_j. That means, an agent looking for bundles that satisfy $d_j^T x_j + p^T w \leq z_j(c_j)$. If $p^T w \geq 0$, the term $p^T w$ is interpreted

as the payment to receive the bundle w. For the opposite, if $p^T w \leq 0$, the term $p^T w$ constitutes the revenue for selling the bundle w.

Accordingly, a rational agent will choose an improving bundle for trading, which leaves his wealth on the same level or better off. The basic market mechanism of Guo et al. (2007) assumes no strategic actions in the bundle selection and pricing. However, strategic actions can be relaxed and Guo et al. (2012) also incorporate strategic factors in their extended BTM. The applied market mechanism in the BLEMS implemented nonstrategic bidding, hence an agents bundle selection can be defined as the following *Bundle Determination Problem (BDP):*

Bundle Determination Problem

$$
\begin{aligned}
\min_{x_j \geq 0, w} \quad & d_j^T x_j + p^T w \\
\text{s.t.} \quad & N_j x_j \leq n_j, \\
& C_j x_j \leq c_j + w
\end{aligned}
\tag{4}
$$

According to Guo et al. (2007), the BDP have either a bounded, or an unbounded solution. A bounded solution is called *limited bundle* $w \in R^m$, whereby an unbounded solution is called *unlimited bundle* $u \in R^m$. Each agent needs to determine a limit price, which indicates the maximum an agent is willing to pay for the bundle. As already explained above, nonstrategic pricing is implemented. Therefore, an agent will always submits a limit price equal to the valuation of the bundle. That is to say, $l(w) = v(w)$.

For a limited bundle w, the value $v(w)$ is defined as:

$$ z_j(c_j) - z_j(c_j + w) = z_j(c_j) - d_j^T x_j. $$

Whereas for a unlimited bundle u, the value $v(u)$ describes the unit-incremental value that u contributes to the objective change. It is defined as:

$$ -d_j^T \hat{x}_j. $$

Potentially, an agent has multiple optimal solutions in his bundle-determination problem. Hence, it is allowed to submit more than one bundle at a time. Old orders and bids of prior rounds without any trades will be treated as open orders and bids, because the valuation of an agent for a bundle is independent of the market price p. It only depends on the respective resource level c_j.

8.2.2 Market Matching Mechanism

The BLEMS applied a synchronous call market where the market prices p are announced periodically by the market dealer. However, the synchronous requirement can be relaxed through the asynchronous market trading environment introduced by Guo et al. (2012) .

The dealer trades on its own account and has also an initial cash endowment e_0 and resource endowment c_0. The sum of all resource endowments needs to fulfill the following equation:

$$\sum_{j=1}^{k} c_j + c_0 \leq c$$

Agents submit sealed bids to the dealer. The dealer maintains an individual order book for each agent. Each individual order book contains two different order sets. On the one hand, the set for limited orders I_j, on the other hand, the set for unlimited orders H_j. If an agent submits orders and no trade is executed, the orders are collected and accumulated for the respective agent. Otherwise, any trade from an agent will clear the respective order book, as it cannot be safely assumed that an old bundle still improves the agents LP given its new resource endowment. To determine the maximal trade surplus the dealer solves the following MMP:

Market Matching Problem

$$
\begin{aligned}
\max_{y_j^i \geq 0, t_j^h \geq 0} \quad & \sum_{j=1}^{k} \left(\sum_{i \in I_j} l_j(w_j^i) y_j^i + \sum_{h \in H_j} l_j(u_j^h) t_j^h \right) \\
\text{s.t.} \quad & \sum_{j=1}^{k} \left(\sum_{i \in I_j} w_j^i y_j^i + \sum_{h \in H_j} u_j^h t_j^h \right) \leq c_0 \\
& \sum_{j=1}^{k} y_j^i \leq 1, \quad j = 1, ..., k
\end{aligned}
\tag{5}
$$

Considering the first set of constraints, if $c_0 = 0$, it is required that any buy amounts be met by sell amounts. On the opposite, if $c_0 \neq 0$, the dealer supplies additional resources from the inventory to meet buys.

The second set of constraints indicate that the market handles limited and unlimited bundles differently. The trades of unlimited bundles in H_j are unrestricted, whereas limited bundles in I_j are restricted. In that case, limited trades are restricted to be convex combinations of bundles in I_j.

The MMP always has a solution, because setting all variables to zero is always a feasible solution. As described by Guo et al. (2007), the market closes when no trade takes place and the market prices remain unchanged. Finally, if the MMP

has a nonzero solution for y_j^{i*}, for $i \in I_j$ and t_j^{h*}, for $h \in H_j$, then agent j will have traded the following:

$$w_j^* = \sum_{i \in I_j} w_j^i y_j^{i*} + \sum_{h \in H_j} u_j^h t_j^{h*}.$$

The market clearing prices $p \in R^m$ are derived from the dual values of the first set of constraints in the MMP (to remember, the concept of duality is explained in section 7.1). Consequently, the settlement price for an agent who have traded resources are the following:

$$p^T w_j^*.$$

In the following, the whole settlement process is summarized and the calculation of the new values of agent j and the dealer is outlined:

Agent j:
$$c_j \leftarrow c_j + w_j^*$$
$$e_j \leftarrow e_j + p^T w_j^*$$

Dealer:
$$c_0 \leftarrow c_0 + \sum_{j=1}^{k} w_j^*$$
$$e_0 \leftarrow e_0 + \sum_{j=1}^{k} p^T w_j^*$$

8.2.3   Summary of BTM

This subsection gives a summarized overview of the BTM, which is based on the equations (1)-(5) and present a belonging flowchart in Figure 8.

Input
> Initial allocations: e_j, c_j, for $j = 1, ..., k$, and
> initial market prices: $p \geq 0$.

Output
> Optimal solutions x_{j*} to the central problem,
> optimal allocations c_{j*}, for $j = 1, ..., k$, and
> optimal market prices p^*.

Step 0
> Set $I_j = \emptyset, H_j = \emptyset$, for $j = 1, ..., k$, and
> the last market-price vector $\pi = 0$.

Step 1

> The dealer saves the current market prices $\pi \leftarrow p$. Each agent solves his BDP and adds new limited bundle orders to I_j and unlimited bundle orders to H_j.

Step 2

> The dealer solves the MMP (w_j^* represents a matched bundle for agent j).

Step 3

> If no nonzero solution exists, this step will be skipped and continued with *Step 4*. If a nonzero solution to the MMP exists and the trades include items from at least one agent and the dealer, the settlement takes place. The dealer publishes the shadow prices p as the new market prices. The new allocations are calculated as follows: $c_j \leftarrow c_j + w_j^*$, for $j = 1, ..., k$ and $c_0 \leftarrow c_0 + \sum_{j=1}^{k} w_j^*$. Further, the new endowments are calculated as follows: $e_j \leftarrow e_j + p^T w_j^*$ for $j = 1, ..., k$ and $e_0 \leftarrow e_0 + \sum_{j=1}^{k} p^T w_j^*$. Reset $I_j = \emptyset$ and $H_j = \emptyset$ for each agent j with $w_j^* \neq 0$. Now, go to *Step 1*.

Step 4

> The dual values of the clearing constraints from the MMP form the new market prices p. If the new market prices $p \neq \pi$, go to *Step 1*, otherwise, stop.
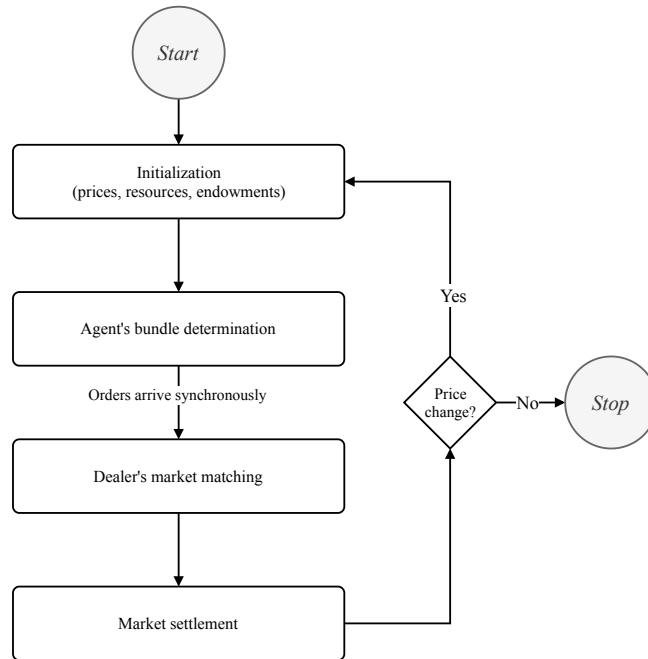

Figure 8: Flowchart of BTM, inspired by Guo et al. (2012)

9   Concept of LEM Simulation Platform

This section brings together the previous sections 6 and 8, and explains how the BTM and a blockchain interact to implement the BLEMS. Finally, this section gives an example of a LP in terms of energy efficient demand side management of households.

First, in the presented BTM in section 8, independent, self-interested agents trading bundled resources in a double-auction market. In case of the BLEMS, the agents of the BTM represent households that trade independently and self-interested bundles of energy resources to minimize their monetary expense through optimally scheduling the operation and energy consumption of all energy consuming appliances.

Second, blockchain is introduced as the applied ICT of the BLEMS. As previously mentioned, the implementation of LEM needs local distributed control and management techniques, which can be addressed by the blockchain technology. This implies that the respective households submit and receive all relevant data, information, and payments via a blockchain. The double-auction market which enables households to trade their energy bundles is operated by a dealer. In turn, the dealer is implemented by a smart contract in conjunction with a conventional software client. The dealer's smart contract contains all relevant information regarding the market mechanism, like submitted orders, trades, dealer's inventory, and market prices. Therefore, households exclusively communicate with the dealer's smart contract.
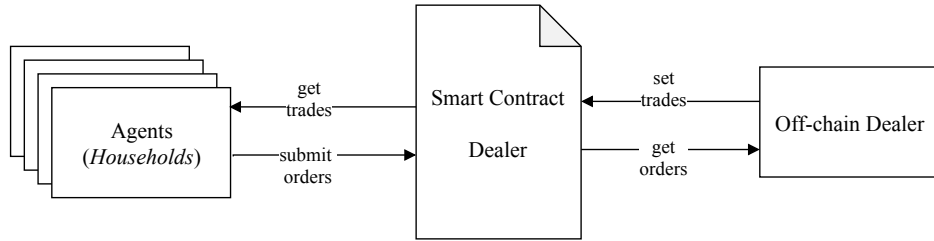
Figure 9: Presentation of the applied communication

As can be seen in Figure 9, the households submit their orders to the dealer's smart contract. Afterwards, the dealer's conventional software client, labeled as *off-chain* dealer in the Figure, fetches all existing orders contained in the smart contract and solves the MMP. The off-chain dealer updates all relevant information in the smart contract, for instance the settled orders, the trades, the new market prices, and the resource inventory. Finally, the households get their respective trades from the dealer's smart contract.

Nevertheless, the off-chain dealer is necessary due to the complex and costly

35

calculation of the MMP. As mentioned in section 6.2.3, each transaction consumes gas. Additionally, each mathematical operation in a smart contract increases the amount of gas to be paid for the respective transaction. As a consequence, implementation and execution of the MMP in the smart contract itself would be hard to realize and associated with high transaction costs.

Hence, a well-designed smart contract should move computational complexity off-chain and focus more on updating state in the contract (*Calculating Costs in Ethereum Contracts - HackerNoon.com*, 2019). A balance between on-chain and off-chain complexity should be found.

The BLEMS applied the synchronous call market, which means that the dealer executes the MMP exclusively after all agents have submitted their orders. We stated that smart contracts are only active and run if they called by a transaction, hence they never run on his own or in the background. For this reason, a mechanism is necessary that triggers the execution of the MMP, for which the implementation of the off-chain dealer is also suitable.

So far, this section presented the basic idea and concept of the BLEMS and explained the necessity of all existing components. In the following section 10, the technical implementation is described in detail.

We stated earlier in this section that the agents of the BTM represent households that trade bundles of energy resources to minimize the monetary expense. In section 8.1, we introduced the individual LPs of the agents. Further, we explained in section 8.2.1 that a rational agent will choose an improving bundle for trading which leaves his wealth level on the same level or better off and defined the selection of such a bundle as the BDP. To embed the BTM into the topic of LEM, we need to define a LP which depicts the energy efficient demand side management of households.

9.1   Demand Side Management of a Household

This subsection embeds the BTM into the topic of LEM and develops an exemplary LP in terms of energy efficient demand side management of households. The objective of the households is to minimize their monetary expense regarding energy consumption depending on several constraints, which ensures the sufficient provision of energy at all times.

Chen et al. (2013) developed such a LP for demand side management of households, on which the following presented model is based.

First, we introduce all relevant variables. Let a denotes an appliance, and A the set of all existing appliances in a household. In that case, an appliance constitutes energy consuming items, e.g. washers, refrigerators, plug-in hybrid

vehicles, etc. The variable x_a denotes the energy consumed by an appliance a to a given point in time. Each appliance $a \in A$ has a maximum energy level that is defined as rated power and described by P_a. It exists a limit on the total energy consumed by various appliances of a household. This total energy consumption limit is described by L_A. In case the total energy consumption limit is exceeded, the power network of the household will be tripped out. Moreover, a household consumes, on the one hand, solar energy from *photovoltaic systems (PV)* and on the other hand energy from the electrical grid. If households do not consume solar energy directly, they have the opportunity to store it in batteries, or export it back to the main power grid if the battery is full. Let e_g denote the energy used from the electrical grid, and e_s the energy produced by the PV. In addition, let p_g and p_s denote the unit price of the energy from the electrical grid and solar energy. We describe the solar energy used by all appliances of a household as y_s, the consumed battery energy as y_b and the remaining solar energy which is not used by the household appliances and stored in the battery as z_b.

Finally, we developed the following LP for optimization of the household monetary expense.

$$
\begin{aligned}
\min \quad & p_g \cdot e_g + p_s \cdot e_s \\
\text{s.t.} \quad & \sum_{a \in A} x_a \leq L_A \\
& x_a \leq P_a, \quad \forall a \in A \\
& \sum_{a \in A} x_a = y_s + y_b + e_g \\
& y_s + z_b \leq e_s
\end{aligned}
\tag{6}
$$

In the table below, all applied variables of the developed LP are listed:

Table 3: Applied variables in LP for optimization of the household monetary expense

Variable	Description
a	energy consuming appliance of household
A	set of all existing energy consuming appliances of household
x_a	consumed energy of an appliance a
P_a	maximum energy level (rated power) of an appliance a
L_A	total energy consumption limit of a household
e_g	energy used from the electrical grid
e_s	energy produced by PV
p_g	unit price of energy from electrical grid
p_s	unit price of energy from PV
y_s	solar energy used by all appliances of a household
y_b	consumed batterie energy by all appliances of a household
z_b	remaining unused solar energy stored in the batterie

As mentioned earlier, the objective of the respective households is to minimize their monetary expense regarding energy consumption. The term $p_g \cdot e_g$ denotes the costs for energy from the electrical grid, whereas the term $p_s \cdot e_s$ denotes the costs for energy from PV. The first constraint ensures that the total energy consumption of all energy consuming appliances of a household does not exceed the limit of L_A. The second constraint describes the requirement that the consumed energy for all existing appliances a in a household is less or equal to the respective rated power P_a. The third constraint makes sure that the energy consumed by all appliances of a household is the sum of the energy from the electrical grid, the solar energy from the PV system and the battery energy. This implies if the amount of solar and battery energy is not sufficient to cover the energy consumption of all appliances, external energy from the electrical grid will be used. Therefore, it is ensured that sufficient energy is available at all times. The fourth constraint describes that the solar energy produced by the PV system is greater or equal to the sum of the consumed and stored solar energy. At times when the battery is fully charged and the energy consumption of the appliances are covered by solar energy, it is possible to sell the surplus of wasted solar energy. Referring to the BTM, the first two constraints of the proposed linear programming model constitutes the independent resources that are managed locally. The shared resources are represented by the energy resources e_g and e_s. Therefore the respective households trade the following limited bundles.

$$w = \begin{pmatrix} e_g \\ e_s \end{pmatrix}$$

Consequently, the objective of the LP is written in matrix-vector notation.

$$\min \quad p^T \cdot w$$

where p^T denotes the transposed price vector.

$$p^T = \begin{pmatrix} p_g & p_s \end{pmatrix}$$

To sum up, this subsection presented a simplified LP for demand side management of a household that can be integrated into the proposed BTM.

The developed model should only serve as an inspiration and example and has no claim to completeness. It is intended to illustrate how the presented framework can be adapted to enable the trading of energy resources in LEM.

10   Implementation

This section outlines the technical implementation of the blockchain-based LEM simulation and presents parts of the developed source code. Therefore, we give a brief introduction to the applied software technologies and present the individual components and their containing attributes and functions. Finally, we present the simulation process that we divided into *Initial Setup* and *Main Simulation Loop*.

10.1   Applied Technologies

In this subsection we give a brief introduction to the applied software technologies and libraries.

Python   An interpreted, object-oriented, high-level programming language with dynamic semantics. Additionally, it is simple and comes with an easy to learn syntax [6]. The programming language of choice, which is used to develop the respective components that communicate via the blockchain.

Web3.py   A Python library for interacting with an Ethereum blockchain [7]. It enables the developed Python components to communicate via the blockchain.

Ganache   A personal blockchain for Ethereum development [8]. It simulates a full client behavior and makes developing Ethereum applications faster, easier and safer. Ganache is provided on the one hand as a software tool with a graphical user interface (GUI) and on the other hand as command-line tool (CLI). In the develop LEM simulation the command line version of Ganache is used.

Solidity   A statically typed, contract-oriented, high-level programming language for implementing smart contracts on the Ethereum blockchain [9]. It is used to implement the smart contract dealer.

---

[6]https://www.python.org/

[7]https://github.com/ethereum/web3.py

[8]https://github.com/trufflesuite/ganache-cli

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

10.2   Components

We present the three main components *Agent*, *Smart Contract Dealer* and *Off-chain Dealer* in this subsection and introduce their attributes and functions.

10.2.1   Agent

The agents are implemented by a Python class and the naming is oriented on the proposed BTM. As stated in section 9, the agents represent the respective households in the BLEMS.

Class Attributes: The class attributes are presented and the associated source code is shown in the following Listing 2.

```python
class Agent(object):

def __init__(self, agent_number, account_address, web3, dealer_contract):
    self._name = 'AGENT{}'.format(agent_number)
    self._account_address = account_address
    self._web3 = web3
    self._dealer_contract = dealer_contract
    self._optimization_problem = None
    self._bundle_set = None
    self._bid = None
    self._mkt_prices = None
    self._trade = None
    self._objective = None
    self._wealth = None
    self._accept_trade = None
```

Listing 2: Class Attributes of Agent

The variables `agent_number`, `account_address`, `web3` and `dealer_contract` are initialized when the class is instantiated. The variable `agent_number` is used to identify the different agents. Due to its uniqueness, the `account_address` could be also used as an identifier, but it would be cumbersome due to the cryptic hexadecimal representation of the blockchain account addresses. Therefore a more comprehensible name for each agent is established. Furthermore, each agent is assigned an Ethereum blockchain account defined by the variable `account_address`. It is required to submit and receive transactions via the blockchain. The Python library for interacting with an Ethereum blockchain is represented by the variable `web3`. Moreover, the `dealer_contract` constitues an object of the dealer's smart contract that knows all of its implemented functions. Hence, this variable is needed to call the functions of the smart contract out of this Python class. All other presented class attributes in the Listing are set during the simulation process.

Next, some of the essential functions of the class are introduced and described in detail.

Setter of Optimization Problem: The agents require a function for receiving their optimization problem, which implementation is presented in Listing 3.

```
@optimization_problem.setter
def optimization_problem(self, value):
    self._optimization_problem = value
    self._objective = self._optimization_problem.solve().fun
    self._wealth = self.balance + abs(self._objective)
```

Listing 3: Setter of Optimization Problem

After setting the optimization problem, the class attributes `objective` and `wealth` are determined. In this case, the attribute `wealth` is defined exactly as in the BTM. The `balance` constitues the cash endowment e_j and `objective` the optimal value of the agents problem depending on the current amount of the shared resources $z_j(c_j)$. However, `balance` is not a class attribute, but a class function that uses `web3` and `account_address` to retrieve the current balance of the account from the blockchain.

Determination of Bundle Attributes: The implementation of the function for solving the BDP and hence determining the improving bundle set and the associated bid is introduced in the Listing 4.

```
def determine_bundle_attributes(self):
    result = solve_bundle_determination(self._optimization_problem, self.↩
        _mkt_prices)
    self._bundle_set = result.x
    self._bid = self._objective - result.fun
```

Listing 4: Determination of Bundle Attribute

First, the BDP is solved in a separate function in subject to the optimization problem and the current market prices. The associated source code of the function can be found in appendix A.

The `bundle_set` constitutes the improving bundle set and is assigned from the results of the BDP. Further, the `bid` is initialized. We stated that the BTM implemented nonstrategic bidding and pricing, which is why an agent always submits a limit price equal to the valuation of the bundle, such that $l(w) = v(w)$. The value of `bid` is assigned by the difference of the `objective` $z_j(c_j)$ and the objective of the optimization problem with the additional resources of the `bundle_set` $z_j(c_j + w)$, which reflects the definition of the bundles true value $v(w) = z_j(c_j) - z_j(c_j + w)$.

Submitting of Order: Listing 5 presents the implementation of the function for submitting orders to the dealer's smart contract. Currently, smart contracts in Ethereum do not fully support fixed point numbers. It is possible to declare them, but they cannot be assigned to or from (*Solidity Documentation*, 2019). This poses a problem because the values in the BTM often contain fixed point numbers. Therefore, we developed a workaround that shifts the decimal places to the right and truncates the remainder when values sent to the dealer's smart contract so that integer values are generated. When obtaining values from the dealer's smart contract we shift the decimal places to the left to restore the original fixed point values. Due to this, the `bundle_set` and `bid` are prepared for sending to the dealer's smart contract.

```python
def set_order(self):
    bundle_set = utils.prepare_for_sending(self._bundle_set)
    bid = utils.prepare_for_sending(self._bid)

    if(bid > 0):
        prepayment = utils.from_ether_to_wei(self._bid)
    else:
        prepayment = 0

    self._dealer_contract.contract.functions.setOrder(bundle_set, bid, ↵
        prepayment).transact({'from': self._account_address, 'value': ↵
        prepayment})
```

Listing 5: Submitting of Order to Smart Contract Dealer

Additionally, it is identified if the `bid` represents a buy or a sell order by examining if the value of `bid` is greater than zero. In case of a buy order, the agents have to pay in advance, to which the variable `prepayment` is assigned in the amount of `bid`. We implemented this procedure to prevent the strategic placing of orders and to ensure that the financial resources are available. In case of a sell order, no payment in advance is required and the `prepayment` is set to zero. As stated earlier, Wei is a subdivision of the cryptographic currency of Ethereum and presents the base unit within smart contracts. As 1 Ether is defined as 10^{18} Wei, we decided to specify all monetary values on the side of the agents and the off-chain dealer in Ether for better clarity. Therefore, the `bid` is converted from Ether to Wei before assigning to the `prepayment`. Finally, the `dealer_contract` is used to call the function of the dealer's smart contract which is responsible for setting the orders.

Agent Trade Verification: Moving calculations off the blockchain comes along with the drawback that source code is not transparent to all agents anymore. That comes to bear in the opacity of the trade calculation of the dealer. To adress this issue we implemented a trade verification procedure based on strong duality theory, which implementation is outlined in Listing 6. The verification procedure

applies the Strong Duality Theorem to verify the validity of the off-chain dealer's calculations. That means, if the primal MMP has an optimal solution x^*, then the dual also has an optimal solution y^*, such that $c^T x^* = b^T y^*$. Due to this, it is possible to verify if the solution of the dealer is optimal, without solving the corresponding MMP. The agents calculate whether strong duality holds and decide on the basis of the result whether they accept the trade or not. Because of the limitations of floating point arithmetic [10], we round off the values of the MMP and the dual problem to make sure that correct trades are not rejected based on limited machine accuracy. Finally, the class variable `accept_trade` is assigned according to the result of the trade verification procedure.

```python
def verify_strong_duality(self):
    mmp_values, mmp_duals, mmp_target_coefs, mmp_bounds = self.↩
        get_mmp_attributes()
    primal_solution = (-np.sum(mmp_values * mmp_target_coefs))
    dual_solution = np.sum(mmp_duals * mmp_bounds)
    primal_solution = math.floor(primal_solution * 10) / 10
    dual_solution = math.floor(dual_solution * 10) / 10

    if primal_solution == dual_solution:
        self._accept_trade = True
    else:
        self._accept_trade = False
```

Listing 6: Agent Trade Verification

Notification of Trade Acceptance: Likewise, the class contains a function for notifing the dealer's smart contract if the respective trade is accepted or not, as shown in the Listing 7.

```python
def accept_trade(self):
    if(self._accept_trade):
        trade = self._dealer_contract.contract.functions.acceptTrade(self.↩
            _accept_trade).call({'from': self._account_address})
        self._trade = utils.prepare_for_storing(trade)

    self._dealer_contract.contract.functions.acceptTrade(self._accept_trade↩
        ).transact({'from': self._account_address})
```

Listing 7: Notification of Trade Acceptance

In case the trade is accepted, it is fetched and stored in the variable `trade`. Since the trade is fetched from the dealer's smart contract, the stated workaround that shifts the decimal places follows at this point to restore the original fixed point values. Furthermore, an almost identical function invocation of the dealer's smart contract function `acceptTrade()` is indicated twice. The difference lies in the methods `call()` and `transact()`. The `call()` method enables to invoke

---

[10]https://docs.python.org/2/tutorial/floatingpoint.html

any smart contract function on a read-only basis and returns values sent by
the smart contract return statement. Those read-only invocations run much
faster than transactions that require network verification. On the contrary, the
`transact()` method submits a verified transaction that potentially changes the
state of the blockchain. Hence, network verification is needed that causes a
significant delay due to the mining procedure. Consequently, this method does
not return the values of the smart contract return statements, but the transaction
hash of the submitted verified transaction. Regarding the implementation, the
first invocation only serves to fetch the respective trade, whereas the second call
of the function serves to notify the dealer whether the trade has been accepted
or not. For this reason, the first call of the dealer's smart contract function
`acceptTrade()` is only executed if the trade is accepted.

Adding of Trade to Shared Resources: The implementation of the function
for adding the received trade to the shared resources is presented in Listing 8.

```python
def add_trade_to_shared_resources(self):
    if(self._accept_trade):
        self._optimization_problem.shared_resources = np.add(self.↩
            _optimization_problem.shared_resources, self._trade)
        self._objective = self._optimization_problem.solve().fun
        self._wealth = self.balance + abs(self._objective)
```

Listing 8: Adding of Trade to Shared Resources

The body of the function is only executed if the trade has been accepted. First,
the new allocation of the shared resources is calculated ($c_j \leftarrow c_j + w_j^*$). Due to
the new allocation of shared resources, the `objective` and also the `wealth` is
recalculated accordingly.

10.2.2   Smart Contract Dealer

The smart contract dealer is implemented by the programming language Solidity,
which is the most popular and frequently used language for Ethereum smart
contracts. In the beginning, the attributes are outlined and presented in Listing
9.

Contract Attributes: First, general attributes are declared. We stated that
the creator of a smart contract does not have any special permissions at the pro-
tocol level, but that it is possible to specify explicitly those special permissions in
the source code. For this reason we initialized the variable `owner` with the address
of the contract creator, who is, in our case, the off-chain dealer. The `owner` comes
to bear in the implementation of the modifier `onlyByOwner()`. A modifier can be

```
contract Dealer{
    // general attributes
    address private _owner;
    int256[] public resource_inventory;
    int256[] public mkt_prices;

    // attributes for order management
    uint32 public order_count;
    uint32[] public order_indices;
    mapping(uint32 => Order) public orders;

    // attributes for trade management
    mapping (address => uint256) account_index;
    address[] public trades_accounts;
    mapping(address => int256[]) public trades;
    mapping(address => uint256) public bills;
    mapping(address => uint256) public prepayments;
    mapping(address => uint256) public refunds;
```

Listing 9: Attributes of Smart Contract Dealer

seen as an extension of a function and is mainly used to check if certain conditions are met before executing the rest of the source code in the body of a function. We have built the modifier to ensure that only the owner of the contract has the authority to perform certain functions. In addition, attributes for managing the orders and trades of the agents are declared. The orders, trades and additional properties of the trades are predominantly managed in mappings. A mapping is a collection of key-value pairs, in which all keys and all values must be of the same data type. However, Solidity does not provide an iteration function to retrieve all the lists that are stored in a mapping, why it is not possible to read the values from the mapping without knowing the respective keys. Therefore, we developed a workaround and store the keys of mappings in additional arrays which serves as look up tables. With reference to the Listing 9, the array `order_indices` functions as a look up table for the mapping `orders`, and the array `trades_accounts` for the mapping `trades` and all other trades related mappings.

Next, the essential functions of the dealer's smart contract are introduced and described in detail.

Receiving of Orders: The function for receiving the orders of the agents is outlined in the Listing 10. It contains, on the one hand, the Solidity built-in modifier `payable`, and on the other hand, the custom modifier `checkPrepayment`. To allow a function to receive payments, it must be declared as `payable`, otherwise the incoming payment would not be accepted and the belonging transaction would not terminate successfully. In order to receive the prepayments of the agents, this function requires the stated modifier. Further, we developed the second modifier `checkPrepayment`, which checks whether the amount sent with the transaction

corresponds to the amount to be paid. Thus it is verified if the transaction is a valid order. Hence, if the prepayment is not sufficient, the function call of the agents is rejected and the order is not accepted.

```
function setOrder(int256[] _bundle, uint256 _bid, uint256 _prepayment) ↩
    public payable checkPrepayment(_prepayment) {

    Order memory new_order = Order(
        msg.sender,
        _bundle,
        _bid
    );
    orders[order_count] = new_order;
    order_indices.push(order_count);

    // increment order count
    order_count ++;
}
```

Listing 10: Receiving of Orders from Agents

If the preconditions defined by the modifiers are fulfilled and the function body is executed, the variable `new_order` is initialized by the object `Order`. The object is implemented by the data type struct, which is a user-defined data container for grouping variables and is well suited to bundle and manage all necessary information regarding the orders. The `msg.sender`, `bundle` and `bid` are contained in the data container and represent the essential information regarding an agents order. When a contract is executed, it has access to a small set of global objects to which the object `msg` belongs. This object contains information regarding the transaction which invokes the contract execution, such as the account address that can be accessed by the variable `msg.sender`. Finally, the `new_order` is added to the mapping `orders`, to which the incremental number `order_count` serves as the key. After each incoming order, the number is increased by one and added to the array `oder_indices` that serves as a look up table as mentioned earlier.

Receiving of Trades: The function which serves to set the trades of the agents is introduced in Listing 11. With reference to the function header, the modifier `onlyByOwner()` is applied, which ensures that only the owner of the contract has the authority to call this function. It is required, since nobody except the owner (off-chain dealer) should have the permission to call this function and thus to set the trades of the agents. Then, the function `addToArray()` is called, which examines whether the account, is already contained in the array `trades_account`. If not, the account is added to the array, which again serves as the look up table for the mapping `trades` and all other trades related mappings.

In conclusion, all the required information for a trade are placed in the corre-

```
    function setTrade(address _account, int256[] _trade, uint256 _prepayment, ↩
        uint256 _bill, uint256 _refund) public onlyByOwner() {
        addToArray(_account);
        trades[_account] = _trade;
        bills[_account] = _bill;
        prepayments[_account] = _prepayment;
        refunds[_account] = _refund;
    }
```

Listing 11: Receiving of Trades from Off-chain Dealer

sponding mappings, to which the account address is used as the key.

Interface of Trades: The function for providing the trades of the agents is
outlined and shown in Listing 12. It can be seen as the interface of trades, as it
enables agents to accept and fetch or to reject their trades.

```
    function acceptTrade(bool accept_trade) public returns (int256[]) {
        if(accept_trade) {
            msg.sender.transfer(refunds[msg.sender]);
            return trades[msg.sender];
        } else {
            msg.sender.transfer(prepayments[msg.sender]);
            delete trades[msg.sender];
        }
    }
```

Listing 12: Interface of Trades

In case of trade acceptance, the trade is returned and the refund is paid to the
respective agent who invokes the function. As stated, the agents have to prepay
the true value of the requested bundle, but they often only get shares of their
requested bundles due to the solution of the MMP. For this reason, the agents
have often prepaid more than they have to pay, which leads to excess payments.
Because of this, the dealer calculates the difference between the prepayment and
the price of the granted trade and refunds it. In case of trade rejection, the dealer
refunds the whole prepayment and deletes the granted trade. The deletion of a
rejected trade is necessary to calculate the dealer's resource inventory correctly
after the settlement process.

10.2.3   Off-chain Dealer

The dealer is implemented by a smart contract in conjunction with a conventional
software client. In contrast to the subsection in which we introduced the Smart
Contract Dealer, this chapter introduces the part of the conventional software
client called Off-chain Dealer, implemented by a Python class.

Class Attributes: The class attributes are described and the associated source
code is provided in Listing 13. Similar to the implementation of the agents, the

`account_address`, `web3`, `dealer_contract` are initialized by instantiation of the class and serve for the same purpose. Besides, the attributes `shared_resource_size` and `mkt_prices` are initialized simultaneously. The `shared_resource_size` constitutes the amount of the various shared resources c and is required to determine the vector size of the traded bundles and related market price vector. The variable `mkt_prices` represents the market price vector and is initialized by zeros with subject to the `shared_resource_size`. The other shown class attributes in the Listing are set during the simulation process.

```python
class Dealer(object):

    def __init__(self, account_address, web3, dealer_contract, ↩
        shared_resource_size):
        self._account_address = account_address
        self._web3 = web3
        self._dealer_contract = dealer_contract
        self._resource_inventory = None
        self._trade = None
        self._shared_resource_size = shared_resource_size
        self._mkt_prices = np.zeros(self._shared_resource_size)
        self._order_handler = None
```

Listing 13: Class Attributes of Off-chain Dealer

Next, the essential functions of the class are introduced and explained in detail.

Fetching of Orders: The function to retrieve the orders of the agents from the dealer's smart contract is introduced in the Listing 14.

```python
def get_orders(self):
    self._order_handler = utils.OrderHandler()
    order_indices = self.get_order_indices()

    # get all orders from contract and store in order handler
    for order_id in order_indices:
        order = self._dealer_contract.contract.functions.getOrder(order_id)↩
            .call()
        self._order_handler.add_order(order_id, order)
```

Listing 14: Fetching of Orders from Smart Contract Dealer

We developed a class `OrderHandler` to store and manage the orders of the agents' more easily on the side of the off-chain dealer. It works like a kind of dictionary and contains the account addresses as the key and the corresponding trades as values, but additionally provides methods to calculate the relevant attributes of the trades in subject to the orders. Therefore, the variable `order_handler` is initialized with an object of `OrderHandler` and the iteratively retrieved orders from the dealer's smart contract are stored in it.

Furthermore, the class also provides functions for creating and solving the MMP. Due to the focus on the architectural design and implementation of the

simulation and the high complexity of these functions, they are not presented in detail. However, the corresponding source code can be found in appendix B.

Setting of Trades: The function to set the trades of the agents in the dealer's smart contract is outlined and presented in the Listing 15. The results from the MMP represent the share of the respective orders and are set for each order in the `order_handler`. Then, the calculation of the relevant attributes of the trades (prepayment, bill, and refund) is executed, whereby the provided methods of the `OrderHandler` come to bear.

```python
def set_trades(self):
    self.set_trade_share()
    self.initiate_trade_calculation()
    self.initiate_prepayment_calculation()
    self.initiate_bill_calculation()
    self.initiate_refund_calculation()

    for order in self._order_handler.get_all_orders():
        account, trade, prepayment, bill, refund = order.↵
            get_trade_information()
        prepayment = utils.from_ether_to_wei(prepayment)
        bill = utils.from_ether_to_wei(bill)
        refund = utils.from_ether_to_wei(refund)

        self._dealer_contract.contract.functions.setTrade(account, trade, ↵
            prepayment, bill, refund).transact({'from': self.↵
            _account_address})
```

Listing 15: Setting of Trades in Smart Contract Dealer

In conclusion, the calculated trades and the corresponding attributes are set iteratively in the dealer's smart contract. Again, the monetary values like `prepayment`, `bill`, and `refund` are converted from Ether to Wei and the values of the trades are converted to integer by shifting the decimal places.

10.2.4   Blockchain

The applied blockchain is described in this subsection. In section 10.1, we already stated that we used a personal blockchain called Ganache, which acts like a personal Ethereum full client. However, Ganache provides even more features that allow you to customize the properties of the Blockchain. First, you can specify the number of generated accounts and the assigned amount of ether. This is a proper way to specify the initial cash endowments of the agents (e_j, for $j = 1, ..., k$) and the dealer (e_0). Further, it is possible to adjust the block time, which defines the time it takes to mine a new block. Additionally, it provides the feature to instantly mine a new block for every transaction. This is very useful for testing and debugging applications like the BLEMS. Likewise, it is possible to specify the price of gas in Wei and the gas limit of a block. Hence, the applied

blockchain Ganache allows you to recreate many scenarios of different blockchain behavior and is therefore perfectly suited for the application of a simulation.

10.3   Simulation Process

This subsection presents the whole simulation process. In contrast to the previous subsection that considered the individual components in isolation, this subsection focus on the interaction of the various components. Therefore, we divided the simulation process into *Initial Setup* and *Simulation Loop*, which are described in the following.

10.3.1   Initial Setup

The part of the initial setup and the associated source code is shown in the Listing 16. As the name suggests, all initial configurations for the simulation are performed in this part.

```
# general setup of simulation
var = mem.Variables()
lp.decompose(var)

# set initial inventory and market prices
var.dealer.set_resource_inventory()
var.dealer.set_mkt_prices()
```

Listing 16: Initial Setup of Simulation

Global Memory: An object of the class `Variables` is instantiated, which can be seen as a kind of global memory. It contains all fundamental variables, which are required to run the simulation. These variables are described in more detail in the following list.

`central_problem:` Represents the specified central optimization problem of the presented BTM. It is divided into subproblems and distributed to the existing agents. It is important to note that the number of variables present in the problem can be exactly divided by the number of agents. Otherwise, an exception is thrown.

`web3:` Represents the object of the Python library, which enables the interaction with an Ethereum blockchain. It is the same object that is passed to the agent classes and the off-chain dealer class.

`dealer_contract:` Represents the object of the dealer's smart contract. It is the same object that is passed to the agent classes and the off-chain dealer class.

It contains all the implemented functions of the dealer's smart contract. When the contract is deployed and thus anchored in the blockchain, a JSON file is created. This file is denoted as the *Contract Application Binary Interface (ABI)*. The ABI is used to instantiate the Python object.

`agent_pool:` Represents a list containing all existing agent objects. Therefore all existing accounts are read from the blockchain and used to instantiate the agents.

`dealer:` Represents the object of the off-chain dealer, which is instantiated with the remaining account.

Decomposition of Central Problem: Next, the function for decomposing the central problem is called. The class object `Variables` is passed to it as a parameter. This is needed to reach the central problem, the agent pool and thus the amount of all existing agents, and the object of the off-chain dealer. Then, the central problem is decomposed into subproblems which are distributed among the agents. The distribution step also includes the distribution of the shared resources. Further, the allocation of the shared resources is implemented in such a way that all parties receive equal shares.

Setting of Initial Values: Furthermore, the off-chain dealer sets the initial values of the resource inventory and the market prices in the dealer's smart contract. As stated before, the initial resource inventory is calculated in the decomposition step of the central problem. We also mentioned earlier in 10.2.3, that the market price vector is initialized when instantiating the off-chain dealer class. The size of the market price vector is equal to the size of the shared resources and is initially specified with zeros.

Finally, a few simulation parameters are set, which are needed to determine the termination of the simulation. These are self-explanatory and not essential for understanding the simulation process, which is why they are not shown in the Listing 16. Afterwards, the simulation loop starts.

10.3.2   Simulation Loop

The source code of the simulation loop is outlined in Listing 17. It represents the main process of the simulation and is implemented by a while loop, which only terminates if the market prices from the previous round are equal to the market prices of the current round. That means, the simulation runs as long as the market prices of the last two rounds differ.

```python
while(not equal_market_prices):

    # agent bidding
    for agent in var.agent_pool:
        agent.get_mkt_prices()   # call
        agent.determine_bundle_attributes()
        agent.set_order()   # transact

    utils.wait_for_new_block(var)

    # dealers market matching mechanism
    var.dealer.get_orders()   # call
    var.dealer.create_mmp()
    var.dealer.solve_mmp()
    var.dealer.set_trades()   # transact
    var.dealer.delete_order()   # transact
    var.dealer.set_mkt_prices()   # transact
    var.dealer.set_mmp_attributes()   # transact

    utils.wait_for_new_block(var)

    # agent trade verification
    for agent in var.agent_pool:
        agent.verify_strong_duality()   # call
        agent.accept_trade()   # transact
        agent.add_trade_to_shared_resources()

    utils.wait_for_new_block(var)

    var.dealer.recalculate_resource_inventory() # transact
```

Listing 17: Main Simulation Loop

Moreover, the behavior of a blockchain differs from that of conventional client-server architecture. In the Ethereum blockchain environment, two different method calls are available. On the one hand, the `transact()` method, and on the other hand, the `call()` method. As already stated, the difference lies in the fact that the `transact()` method requires the submitting of verified transactions that potentially change the state of the Blockchain, whereas the `call()` method only reads the state of the Blockchain. For a `transact()` method network verification is needed that causes a significant delay due to the mining procedure. Consequently, before the values written into the Blockchain by a `transact()` method are accessible, a certain time passes. It is important to take this into account when trying to read certain values from the Blockchain that are set in previous steps. Due to this, we implemented a function called `wait_for_new_block()` that reads the current number of blocks out of the Blockchain and waits for the generation of a further block. This avoids that state changing transactions are still pending and not anchored in a block while they are requested.

Further, we divided the simulation loop logically into three thematically summarized actions *Agent Bidding*, *Market Matching Mechanism*, and *Agent Trade Verification*.

Agent bidding: The *Agent Bidding* represents the technical implementation of section 8.2.1. Figure 10 outlines the sequence of all function calls that belong to this action.
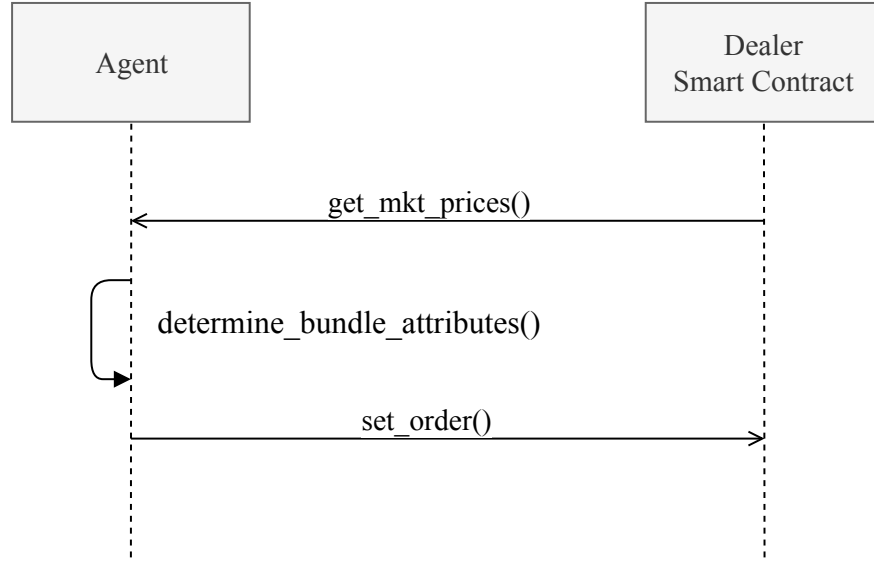

Figure 10: Sequence of Agent Bidding

The agents get the current market prices from the dealer's smart contract, which are required to solve the BDP. To receive the market prices the `call()` method is used and hence proceeds very quickly, as the values are only read from the blockchain. Next, the agents solve the BDP based on the received market prices and submit the orders and the corresponding prepayment to the dealer's smart contract. In contrast to receiving market prices, submitting the orders require the execution of verified transactions that potentially change the state of the blockchain. Therefore, we make use of the stated `wait_for_new_block()` function after the placement of the orders to ensure that all submitted orders are take part in the following *Market Matching Mechanism*.

Market Matching Mechanism: The *Market Matching Mechanism* represents the technical implementation of section 8.2.2 and Figure 11 provide a sequence of all function calls for this action.

The off-chain dealer waits for the anchoring of the submitted orders and fetches the respective bundle orders of the agents. As the off-chain dealer only has to read the values of the orders from the blockchain, fetching the orders is also implemented by a `call()` method, why it proceeds very quickly. In subject to the received orders, the off-chain dealer creates and solves the MMP. Since the MMP is solved off-chain, the variety of implementations to solve the MMP is not limited. Therefore it is possible to apply commercial solvers for improving performance when large problems have to be solved.
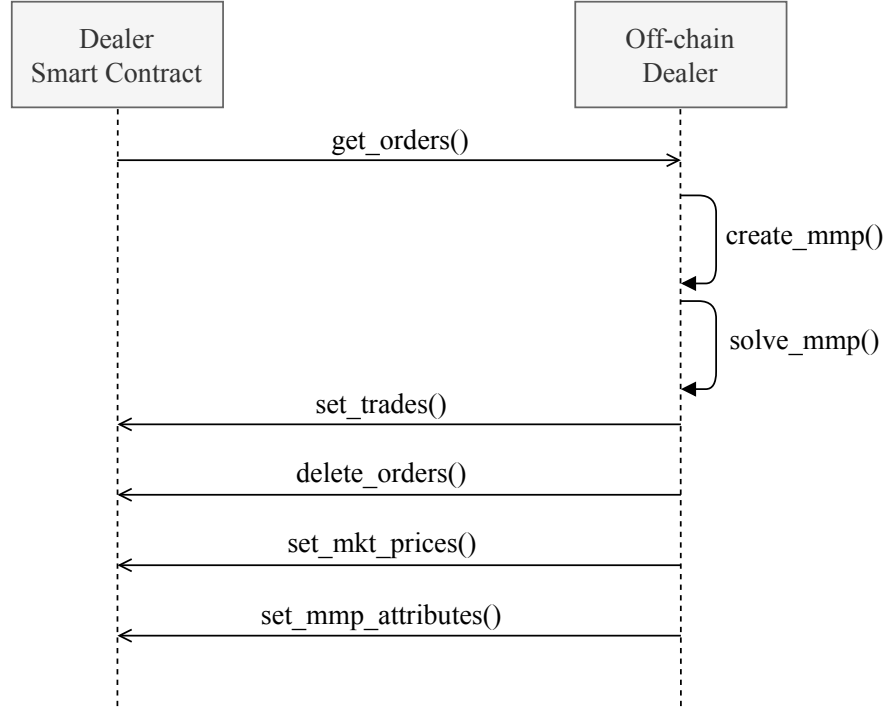
Figure 11: Sequence of Market Matching Mechanism

After solving the MMP and the calculation of the respective trades, the off-chain dealer sets the trades in the dealer's smart contract and deletes the settled orders out of it. The newly derived market prices and the values of the MMP and its duals, which are needed for the agents trade verification, are set in the dealer's smart contract. Likewise, the setting of values in the Smart Contract also requires verified transactions that have been implemented using the `transact()` method. Since the values are required for the agents trade verification, we again utilize the stated `wait_for_new_block()` function to ensure the anchoring of the values in the blockchain before performing the next action.

Agent Trade Verification: The *Agent Trade Verification* presents the developed procedure to verify the calculated trades of the dealer, which is necessary due to moving the calculation of the MMP off-chain. A sequence of all function calls contained in this action is shown in Figure 12.

The agents verify whether the strong duality theory holds in subject to the results of the MMP and the corresponding dual, which are fetched from the dealer's smart contract. After determining whether the agents accept or reject the trades, the dealer's smart contract is notified. In case of acceptance, the dealer's smart contract transfers the refunds to the agents and returns the respective trades. Otherwise, in case of rejection, the dealer's smart contract transfer the whole prepayment and deletes the respective trades. Finally, the agents add the trades to the shared resources and calculate the new allocation of the shared resources.
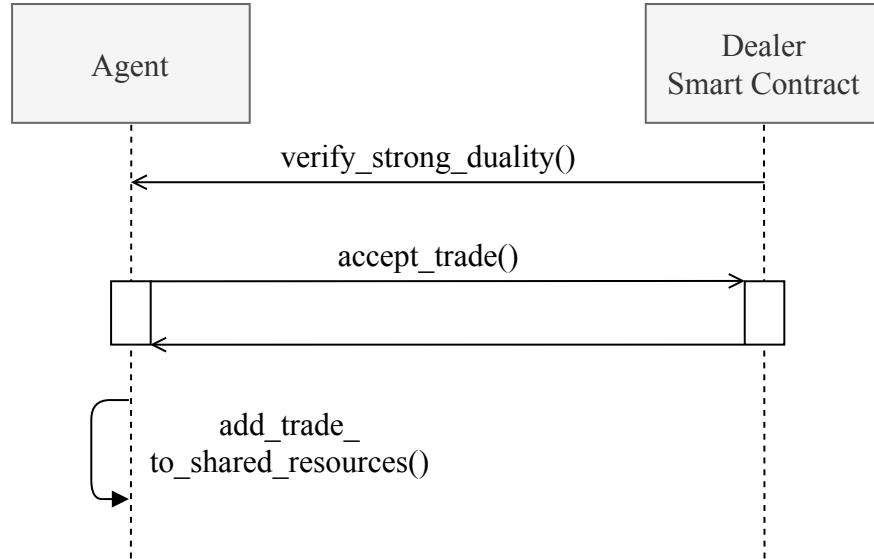
Figure 12: Sequence of Agents Trade Verification

Due to the new allocation of shared resources, the new objective and wealth level of the agents is calculated. The invocation of this function only affects if the trade was previously accepted.

At the end of each iteration, the resource inventory of the dealer is recalculated. Therefore the off-chain dealer calculates how much of the inventory is used for the trades and settles this with the previous inventory. Finally, the new calculated inventory is set in the dealer's smart contract.

11   Conclusion

In this research, we described the difficulties of integrating distributes renewable energy sources (RES) into the existing electric grid and presented peer-to-peer (P2P) energy trading in local energy markets (LEM) as a possible solution to technical and market problems. Further, we pointed out that central management in P2P energy trading is challenging due to the need for advanced communication and data exchanges, which is why local distributed control and management techniques are more suitable. Therefore, we introduced the Ethereum-based blockchain as a new and innovative information communication technology (ICT), which fulfills the need for distributed control and management techniques. Moreover, we presented the Bundle Trading Market Framework (BTM), which solves a distributed system optimization problem by self-interested agents that iteratively trading bundled resources in a double auction market run by a dealer. Then, we brought all introduced topics together and proposed a concept for an open blockchain-based LEM simulation (BLEMS). Therefore, we presented the architectural design and embedded the BTM into the topic of LEM by conceptualizing a linear program (LP) for demand side management of households. Finally, the technical implementation of the BLEMS is outlined, in which we described the issues due to specific properties of blockchain and presented the applied solutions.

   In the following section, we investigate whether the developed simulation contains all components for the efficient operation of a blockchain-based LEM, which are introduced in section 5.1. Then, we outline the contribution of this research. Finally, we examine further areas of research.

11.1   Compliance of LEM Components

In this subsection, we examine if the seven components for the efficient operation of LEM, elaborated by (Mengelkamp, Gärttner, et al., 2018), can be provided by the developed simulation platform. Referring to the microgrid setup (C1), an explicit objective, a definition of the market participants and a definition of the form of the traded energy is required. The developed simulation can meet these requirements. The explicit objective can be defined by the researcher itself by setting up a central problem. It is possible to define the market participants and the form of the traded energy bundles according to your requirements. Referring to the grid connection (C2), well defined connection points to the superordinate main grid are required. These connection points are not implemented yet, but the implementation of these connection points into dealer's smart contract is possible. It would be conceivable to integrate these into the inventory policy of the dealer. As for the information system (C3), the requirements are fulfilled due the usage

of blockchain as the ICT. Moreover, the applied BTM constitutes an appropriate market mechanism (C4). The BTM defines a clear bidding strategy and the overall objective of the applied framework is to provide an optimal allocation in near real-time. Additionally, the BTM determines the respective prices by calculating the dual variables. Therefore, the developed simulation also includes a pricing mechanism (C5) that supports an efficient allocation. Likewise, a suitable EMTS (C6) is provided by the applied BTM. It ensures automatically the energy supply and aims to minimize energy costs. Finally, the developed simulation lacks the definition of how LEM fit into the current energy policy and how taxes and fees are distributed and billed. That is why the component regulation (C7) is not fulfilled. In conclusion, the developed simulation provides five of seven components for the efficient operation of LEM and allows the integration of the missing two components. To remember, the components have no general validity but allow an initial assessment of the platform and its capabilities. Therefore, the BLEMS provides an appropriate and valuable basis for a significant blockchain-based simulation of LEM.

11.2   Contribution

The conducted research provides two core contributions.

First, the decentralized implementation of the proposed BTM. We successfully implemented the BTM with a blockchain as the applied ICT, in which the sum of the agents objectives equals the objective of the central problem. We also applied the numerical example of Guo et al. (2007) and ended up with the same objectives and resource allocations, listed in the log of the executed simulation in appendix C. Since the application of blockchain technology, the technical implementation is publicly accessible, hence more transparency is provided and the required level of trust in the BTM is reduced. Nevertheless, to increase efficiency of the distributed solution, we moved calculations off-chain, which is why the respective source code is not transparent to all stakeholders anymore. To address this issue, we developed the *Agent Trade Verification* procedure, which based on strong duality theory. Therefore, the level of transparency is maintained and the appliance of commercial solvers is enabled. Additionally, the blockchain contains cryptographic encryption methods, which also raises the level of security in the BTM. In conclusion, a secure and decentralized market-based optimization algorithm for distributed systems has been developed.

Second, we conceptualized a blockchain-based LEM simulation based on the implemented BTM. We proposed an exemplary linear programming problem in terms of energy efficient demand side management of households to embed the

BTM into the topic of LEM. Moreover, the conducted research removes technical barriers. The section 10 provides a detailed explanation of the implementation and deals with the special features of the blockchain functionality. Hence, the usage of complex distributed ledger technologies for researchers is simplified. In addition, the programming language Python is very friendly for beginners and eases the entry into the use of the BLEMS. Therefore, it incentivizes researchers to design and test their artifacts by using the BLEMS. Furthermore, due to the customizability of the blockchain properties the developed simulation allows for testing different scenarios and thus to get a better understanding of the dynamics of a blockchain-based decentralized LEM.

11.3 Future Work

The conducted research offers the potential for future work in many areas.

First, we stated in section 8.2.2, that we applied a synchronous call market, where submitted orders to the system will be accumulated and processed simultaneously at periodic intervals. In a real-world application, it would be more appropriate to clear the market every time new orders arrive. Therefore, the implementation from the synchronous to the introduced asynchronous market trading environment of Guo et al. (2012) would be reasonable.

Second, the implementation of an auction mechanism by a blockchain causes difficulties regarding privacy. Every on-chain data is publicly available. In case of an auction mechanism, this can result in competitive advantages for those who submit their orders later on. This would not be an issue in an environment where all orders submitted simultaneously and written into the same block. Otherwise, in an environment where orders submitted at different times and written into different blocks, some participants will have a competitive advantage, as they can see the orders of others before they have submitted their own order. In a real-world application, this scenario is more likely, hence future work could address the implementation of commitment schemes in the area of agent bidding to avoid those issues.

Finally, as described in this research, we implemented no strategic actions in the bundle selection and pricing. However, Guo et al. (2012) represent various forms of agent strategic behavior regarding bundle selection and bundle pricing. In their proposals, agents use forecasted market prices to determine the preferred bundles instead of the most recently observed market prices. These approaches could be adopted by future work and implemented into the presented BLEMS. In addition, Guo et al. (2012) also introduce two different inventory policies of the dealer to examine if holding intertemporal inventory can facilitate real-time

trades in the BTM environment. The implementation of those inventory policies into the presented simulation and the adaption of these policies into the context of LEM could be also of interest for future work.

A Additional Agent Implementation

```python
def solve_bundle_determination(optimization_problem, mkt_prices):
    bundle_target_coefs = np.concatenate((optimization_problem.target_coefs↩
        , mkt_prices))

    bundle_individual_coefs = np.concatenate((optimization_problem.↩
        individual_coefs, np.zeros(optimization_problem.individual_coefs.↩
        shape)), axis=1)
    bundle_shared_coefs = np.concatenate((optimization_problem.shared_coefs↩
        , np.identity(mkt_prices.size, dtype=float) * (-1)), axis=1)

    bundle_var_geq_zero_constraint = np.concatenate((np.identity(mkt_prices↩
        .size) * (-1), np.zeros(optimization_problem.shared_coefs.shape)), ↩
        axis=1)
    bundle_var_geq_zero_bound = np.zeros(mkt_prices.size)

    bundle_coefs = np.concatenate((bundle_individual_coefs, ↩
        bundle_shared_coefs, bundle_var_geq_zero_constraint))
    bundle_resources = np.concatenate((optimization_problem.↩
        individual_resources, optimization_problem.shared_resources, ↩
        bundle_var_geq_zero_bound))

    result = linprog(bundle_target_coefs, bundle_coefs, bundle_resources)
    result.x = result.x[- mkt_prices.size:]  # remove not bundle ↩
        coefficients

    return result
```

Listing A.1: Solving of Bundle Determination

B   Additional Off-chain Dealer Implementation

```python
def create_mmp(self):
    bundles = [order.get_concatenated_bundles() for order in self.↵
        _order_handler.get_all_orders()]
    bids = [order.get_concatenated_bids() for order in self._order_handler.↵
        get_all_orders()]

    try:
        TARGET_COEFS = np.hstack(bids) * (-1)  # create target coef vector

        self._mmp_amount_variables = np.size(TARGET_COEFS)  # set amount of↵
            variables
        mmp_coefs = np.hstack(bundles)
        var_leq_one_coefs = np.identity(self._mmp_amount_variables, dtype=↵
            float)  # create constraint matrix for y<=1
        var_geq_zero_coefs = np.identity(self._mmp_amount_variables, dtype=↵
            float) * (-1)  # create constraint matrix for y>=0
        mmp_bounds = self._resource_inventory
        var_leq_one_bounds = np.ones(self._mmp_amount_variables, dtype=↵
            float)
        var_geq_zero_bounds = np.zeros(self._mmp_amount_variables, dtype=↵
            float)

        CONSTRAINT_COEFS = np.concatenate((mmp_coefs, var_leq_one_coefs, ↵
            var_geq_zero_coefs), axis=0)  # create final constraint matrix
        CONSTRAINT_BOUNDS = np.concatenate((mmp_bounds, var_leq_one_bounds,↵
            var_geq_zero_bounds))  # create final bounds matrix

        self._mmp_constraint_coefs = CONSTRAINT_COEFS
        self._mmp_constraint_bounds = CONSTRAINT_BOUNDS
        self._mmp_target_coefs = TARGET_COEFS

    except ValueError as error:
        print('Creation of MMP failed!')
        print(error)
```

Listing B.2: Creation of MMP

```python
def solve_mmp(self):
    solvers.options['show_progress'] = False
    sol = solvers.lp(matrix(self._mmp_target_coefs), matrix(self.↵
        _mmp_constraint_coefs), matrix(self._mmp_constraint_bounds))
    self._mmp_values = np.array([float('%.2f' % (sol['x'][i])) for i in ↵
        range(self._mmp_amount_variables)])
    self._mmp_duals = np.array([float('%.2f' % entry) for entry in sol['z'↵
        ]])
    self._mkt_prices = np.array([float('%.2f' % (sol['z'][i])) for i in ↵
        range(self._shared_resource_size)])
```

Listing B.3: Solving of MMP

C   Log of Simulation Output

```
----------------------------------
CENTRAL LP
Target Coefficients:
[-1 -2 -1 -3]
Individual Coefficients:
[[2 1 0 0]
 [0 0 2 3]]
Individual Resources:
[4 9]
Shared Coefficients:
[[1 3 2 1]
 [1 1 1 1]]
Shared Resources:
[8 5]
----------------------------------
INITIAL SETUP

DEALER
account: 0xED3bCC2F9024aa1720e4956672eE484E63cAC418
dealer inventory: [2.67 1.67]
dealer trade: None
market price: [0. 0.]


AGENT1
account: 0x5ad1079fe9252e637BF4091fEAA52eEF25DfB6Cd
balance: 100.0 ether
objective: -2.17
wealth: 102.17
order: None
bid: None ether
trade: None
trade accepted: None
allocation: [2.67 1.67]


AGENT2
account: 0x037b0Bf2463BA87B2D28Ba568Bb49E5400a8de60
balance: 100.0 ether
objective: -5.01
wealth: 105.01
order: None
bid: None ether
trade: None
trade accepted: None
allocation: [2.67 1.67]


----------------------------------
ITERATION 1

DEALER
account: 0xED3bCC2F9024aa1720e4956672eE484E63cAC418
dealer inventory: [ 0.95 -0.  ]
dealer trade: [1.72 1.67]
market price: [0.  2.5]
```

```
AGENT1
account: 0x5ad1079fe9252e637BF4091fEAA52eEF25DfB6Cd
balance: 99.1220459 ether
objective: -3.035
wealth: 102.1570459
order: [9.33  2.33]
bid: 5.830000000000002 ether
trade: [1.39  0.34]
trade accepted: True
allocation: [4.06  2.01]


AGENT2
account: 0x037b0Bf2463BA87B2D28Ba568Bb49E5400a8de60
balance: 96.67139874 ether
objective: -9.0
wealth: 105.67139874
order: [0.33  1.33]
bid: 3.99 ether
trade: [0.33  1.33]
trade accepted: True
allocation: [3.  3.]


----------------------------------
ITERATION 2

DEALER
account: 0xED3bCC2F9024aa1720e4956672eE484E63cAC418
dealer inventory: [0.01  0.   ]
dealer trade: [0.94  0.   ]
market price: [0.5   0.49]


AGENT1
account: 0x5ad1079fe9252e637BF4091fEAA52eEF25DfB6Cd
balance: 98.64304464 ether
objective: -3.505
wealth: 102.14804464
order: [1.97  0.   ]
bid: 0.9849999999999994 ether
trade: [0.94  0.   ]
trade accepted: True
allocation: [5.    2.01]


AGENT2
account: 0x037b0Bf2463BA87B2D28Ba568Bb49E5400a8de60
balance: 96.66884428 ether
objective: -9.0
wealth: 105.66884428
order: [0.  0.]
bid: 0.0 ether
trade: [0.  0.]
trade accepted: True
allocation: [3.  3.]


----------------------------------
```

```
ITERATION 3

DEALER
account: 0xED3bCC2F9024aa1720e4956672eE484E63cAC418
dealer inventory: [0.01 0.  ]
dealer trade: [0. 0.]
market price: [0.   0.48]


AGENT1
account: 0x5ad1079fe9252e637BF4091fEAA52eEF25DfB6Cd
balance: 98.6193761 ether
objective: −3.505
wealth: 102.12437609999999
order: [7.   1.99]
bid: 0.019899999999999807 ether
trade: [0. 0.]
trade accepted: True
allocation: [5.   2.01]


AGENT2
account: 0x037b0Bf2463BA87B2D28Ba568Bb49E5400a8de60
balance: 96.66628982 ether
objective: −9.0
wealth: 105.66628982
order: [0. 0.]
bid: 0.0 ether
trade: [0. 0.]
trade accepted: True
allocation: [3. 3.]


----------------------------------
ITERATION 4

DEALER
account: 0xED3bCC2F9024aa1720e4956672eE484E63cAC418
dealer inventory: [0.01 0.  ]
dealer trade: [0. 0.]
market price: [0.   1.96]


AGENT1
account: 0x5ad1079fe9252e637BF4091fEAA52eEF25DfB6Cd
balance: 95.076105 ether
objective: −3.505
wealth: 98.581105
order: [7.   1.99]
bid: 3.5397999999999996 ether
trade: [0. 0.]
trade accepted: True
allocation: [5.   2.01]


AGENT2
account: 0x037b0Bf2463BA87B2D28Ba568Bb49E5400a8de60
balance: 96.66373536 ether
objective: −9.0
wealth: 105.66373536
```

```
order: [0.  0.]
bid: 0.0 ether
trade: [0.  0.]
trade accepted: True
allocation: [3.  3.]


----------------------------------
ITERATION 5

DEALER
account: 0xED3bCC2F9024aa1720e4956672eE484E63cAC418
dealer inventory: [0.01 0.   ]
dealer trade: [0.  0.]
market price: [0.    1.96]


AGENT1
account: 0x5ad1079fe9252e637BF4091fEAA52eEF25DfB6Cd
balance: 94.47773618 ether
objective: −3.505
wealth: 97.98273617999999
order: [7.    1.99]
bid: 0.5945999999999998 ether
trade: [0.  0.]
trade accepted: True
allocation: [5.    2.01]


AGENT2
account: 0x037b0Bf2463BA87B2D28Ba568Bb49E5400a8de60
balance: 96.6611809 ether
objective: −9.0
wealth: 105.6611809
order: [0.  0.]
bid: 0.0 ether
trade: [0.  0.]
trade accepted: True
allocation: [3.  3.]
```

D   Installation Guide

Above all, you can find the repository of the developed simulation under the following link:
`https://github.com/nreinhol/lem_sim`

   To start, open your terminal and go into the `lem_sim` directory by typping the following expression:
- `cd /path/to/lem_sim`

1. First of all, we have to start the local blockchain. For this type in:
   - `docker-compose up ganache`

2. After that, we need to deploy the smart contract into the running local blockchain. Therefore, we open another terminal window and change the directory to:
   - `cd /path/to/lem_sim/dealer`

3. To deploy the smart contract, type in the following expression:
   - `truffle migrate --network development`

4. After these steps, the blockchain and the smart contract are ready for communication. Now, we move to the python application. Therefore, change the directory to:
   - `cd path/to/lem_sim/pyapp`

5. Now, you need to set up the python virtual environment. Therefore, type in the following expression:
   - `make requirements`

6. This expression will create a virtual environment in the folder "'pyapp/env"' and install all required python packages into this virtual environment. After the successful set up, type in the following expression to run the python simulation:
   - `make simulate`

Hint: To redeploy contracts without making any changes in the code base, use the following command:
- `truffle migrate --reset --network development`
Otherwise, you have to restart the local blockchain and deploy the contract.

References

Aitzhan, N. Z., & Svetinovic, D. (2018). Security and privacy in decentralized energy trading through multi-signatures, blockchain and anonymous messaging streams. *IEEE Transactions on Dependable and Secure Computing*, *15*(5), 840–852.

Al Kawasmi, E., Arnautovic, E., & Svetinovic, D. (2015). Bitcoin-based decentralized carbon emissions trading infrastructure model. *Systems Engineering*, *18*(2), 115–130.

Ampatzis, M., Nguyen, P. H., & Kling, W. (2014). Local electricity market design for the coordination of distributed energy resources at district level. In *Innovative smart grid technologies conference europe (isgt-europe), 2014 ieee pes* (pp. 1–6).

Andoni, M., Robu, V., Flynn, D., Abram, S., Geach, D., Jenkins, D., . . . Peacock, A. (2019). Blockchain technology in the energy sector: A systematic review of challenges and opportunities. *Renewable and Sustainable Energy Reviews*, *100*, 143–174.

Antonopoulos, A. M., & Wood, G. (2018). *Mastering ethereum: building smart contracts and dapps.* O'Reilly Media.

Bertsekas, D. P. (1992). Auction algorithms for network flow problems: A tutorial introduction. *Computational optimization and applications*, *1*(1), 7–66.

Bichler, M. (2017). *Market design: A linear programming approach to auctions and matching.* Cambridge University Press.

Buterin, V., et al. (2013). A next-generation smart contract and decentralized application platform. *Ethereum White Paper*.

Calculating Costs in Ethereum Contracts - HackerNoon.com. (2019). Retrieved 2019-07-10, from `https://hackernoon.com/ether-purchase-power-df40a38c5a2f`

Chen, X., Wei, T., Hu, S., & Member, S. (2013). Uncertainty-Aware Household Appliance Scheduling Considering Dynamic Electricity Pricing in Smart Home. *IEEE TRANSACTIONS ON SMART GRID*, *4*(2). Retrieved from `http://ieeexplore.ieee.org`. doi: 10.1109/TSG.2012.2226065

Clearwater, S. H. (1996). *Market-based control: A paradigm for distributed resource allocation.* World Scientific.

Dantzig, G. B., & Wolfe, P. (1961). The decomposition algorithm for linear programs. *Econometrica (pre-1986)*, *29*(4), 767–778.

Fan, M., Stallaert, J., & Whinston, A. B. (2003). Decentralized mechanism design for supply chain organizations using an auction market. *Information Systems Research*, *14*(1), 1–22.

Griva, I., Nash, S. G., & Sofer, A. (2009). *Linear and nonlinear optimization* (Vol. 108). Siam.

Guo, Z., Koehler, G. J., & Whinston, A. B. (2007). A market-based optimization algorithm for distributed systems. *Management Science*, *53*(8), 1345–1358.

Guo, Z., Koehler, G. J., & Whinston, A. B. (2012). A computational analysis of bundle trading markets design for distributed resource allocation. *Information Systems Research*, *23*(3-part-1), 823–843.

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2008). Design science in information systems research. *Management Information Systems Quarterly*, *28*(1), 6.

Jimeno, J., Anduaga, J., Oyarzabal, J., & de Muro, A. G. (2011). Architecture of a microgrid energy management system. *European Transactions on Electrical Power*, *21*(2), 1142–1158.

Ketter, W., Peters, M., Collins, J., & Gupta, A. (2015). Competitive benchmarking: an is research approach to address wicked problems with big data and analytics.

Light-Client-Protocol. (2019). Retrieved 2019-03-28, from `https://github.com/ethereum/wiki/wiki/Light-client-protocol`

Long, C., Wu, J., Zhang, C., Cheng, M., & Al-Wakeel, A. (2017). Feasibility of peer-to-peer energy trading in low voltage electrical distribution networks. *Energy Procedia*, *105*, 2227–2232.

Luenberger, D. G. (2008). *Linear and Nonlinear Programming 4TH* (Tech. Rep.). Retrieved from `http://www.springer.com/series/6161`

March, S. T., & Smith, G. F. (1995). Design and natural science research on information technology. *Decision support systems*, *15*(4), 251–266.

Medium, Ethereum-Clients. (2019). Retrieved 2019-03-28, from `https://medium.com/@eth.anBennett/ethereum-clients-101-beginner-geth-parity-full-node-light-client-4bbd87bf1dee`

Medium, Life Cycle of Transactions. (2017). Retrieved 2019-03-14, from `https://medium.com/blockchannel/life-cycle-of-an-ethereum-transaction-e5c66bae0f6e`

Mengelkamp, E., Gärttner, J., Rock, K., Kessler, S., Orsini, L., & Weinhardt, C. (2018). Designing microgrid energy markets: A case study: The brooklyn microgrid. *Applied Energy*, *210*, 870–880.

Mengelkamp, E., Notheisen, B., Beer, C., Dauer, D., & Weinhardt, C. (2018). A blockchain-based smart grid: towards sustainable local energy markets. *Computer Science-Research and Development*, *33*, 207–214.

Merkling in Ethereum. (2015). Retrieved 2019-02-10, from `https://blog.ethereum.org/2015/11/15/merkling-in-ethereum/`

Mihaylov, M., Jurado, S., Avellana, N., Van Moffaert, K., de Abril, I. M., & Nowé, A. (2014). Nrgcoin: Virtual currency for trading of renewable energy in smart grids. In *European energy market (eem), 2014 11th international conference on the* (pp. 1–6).

Modified Merkle Patricia Trie Specification. (2015). Retrieved 2019-02-07, from `https://github.com/ethereum/wiki/wiki/Patricia-Tree`

Proof of Work. (2019). Retrieved 2019-03-17, from `https://en.bitcoin.it/wiki/Proof_of_work`

Sikorski, J. J., Haughton, J., & Kraft, M. (2017). Blockchain technology in the chemical industry: Machine-to-machine electricity market. *Applied Energy*, *195*, 234–246.

Solidity Documentation. (2019). Retrieved 2019-03-17, from `https://solidity.readthedocs.io/en/develop/types.html#fixed-point-numbers`

Vanderbei, R. J., et al. (2015). *Linear programming.* Springer.

Vukolić, M. (2015). The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In *International workshop on open problems in network security* (pp. 112–125).

Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, *151*, 1–32.

Zhang, C., Wu, J., Long, C., & Cheng, M. (2017). Review of existing peer-to-peer energy trading projects. *Energy Procedia*, *105*, 2563–2568.

Zheng, Z., Xie, S., Dai, H., Chen, X., & Wang, H. (2017). An overview of blockchain technology: Architecture, consensus, and future trends. In *Big data (bigdata congress), 2017 ieee international congress on* (pp. 557–564).

Zheng, Z., Xie, S., Dai, H.-N., & Wang, H. (2016). Blockchain challenges and opportunities: A survey. *Work Pap.–2016*.