Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Jan Olav Moltu

Utilizing the IOTA Smart Contract Platform in a Local Flexibility Market

Master's thesis in Computer Science
Supervisor: John Krogstie
March 2023

NTNU
Norwegian University of
Science and Technology

Jan Olav Moltu

Utilizing the IOTA Smart Contract Platform in a Local Flexibility Market

Master's thesis in Computer Science
Supervisor: John Krogstie
March 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Jan Olav Moltu

Utilizing the IOTA Smart Contract Platform in a Local Flexibility Market

Master Thesis, Spring 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

As renewable energy sources become more prevalent and distributed energy systems gain wider adoption, the demand for efficient and secure energy trading mechanisms grows. Distributed Ledger Technologies (DLTs) hold great potential in facilitating the integration of distributed energy resources into energy markets. However, the shift from a centralized energy system to a decentralized one presents new challenges for scalability, making it crucial for DLTs to be able to scale accordingly.

While much research has been conducted on blockchain-based DLTs, this thesis focuses on the Directed Acyclic Graph (DAG)-based IOTA platform and its newly implemented smart contract protocol. In 2021, the IOTA Smart Contract Platform (ISCP) was introduced, enabling the operation of multiple blockchains on top of the ”Tangle” and potentially increasing the number of transactions.

Despite the potential of the ISCP in energy markets, few existing studies evaluate its performance. The majority of performance assessments is based on the previous version of IOTA and, as such, is out of date and does not factor in the capabilities of ISCP. The thesis presents one of the first assessments of the scalability of ISCP in relation to energy markets.

Using a design science methodology, this thesis conducted three iterative design cycles, resulting in the creation and evaluation of three artifacts. These artifacts leverage smart contracts to support three key functions of the energy market: information storage, trade matching, and contract settlement. By designing and testing these artifacts, the thesis contributes to the development of scalable DLT-based solutions for energy trading in decentralized energy systems.

The results of this study indicate that the default node configuration was able to handle 30 bids per second, and that dividing the network into smaller sub-networks is crucial for maintaining the performance and scalability of DLT energy markets. This is in line with the current throughput of the Ethereum blockchain, but the ISCP offers the added advantage of allowing small local energy markets to operate concurrently and anchor to the global ledger. With the implemented matching algorithm, the maximum number of bids per auction round was between 480 and 540. Depending on variables such as the number of nodes in the network and the hardware the nodes were operating on, the total execution time to confirm a transaction ranged from 4.56 to 15.1 seconds. Although this finding is consistent with other studies related to DLT and the requirements of the energy markets, it falls short of the performance offered by other second-layer solutions.

This study lays the groundwork for future investigations into using the ISCP framework in energy markets. Further research is needed to evaluate the performance of the wasp chain under various conditions, investigate the potential benefits of implementing DLT in centralized energy markets, and explore the possibility of a decentralized estimation system with multiple nodes for increased resilience.

Sammendrag

Etter hvert som fornybare energikilder blir mer utbredt og distribuerte energisystemer blir mer omfattende, øker etterspørselen etter effektive og sikre energihandelsmekanismer. Distributed Ledger Technologies (DLT) har stort potensial for å lette integreringen av distribuerte energiressurser i energimarkedene. Men skiftet fra et sentralisert energisystem til et desentralisert presenterer nye utfordringer for skalerbarhet, noe som gjør det avgjørende for DLTs å kunne skalere tilsvarende.

Mens mye forskning har blitt utført på blockchain-baserte DLTs, fokuserer denne oppgaven på den Directed Acyclic Graph (DAG)-baserte IOTA-plattformen og dens nylig implementerte smart kontraktprotokoll. I 2021 ble IOTA Smart Contract platform (ISCP) introdusert, noe som muliggjør drift av flere blockchains på toppen av "Tangle" og potensielt øker antall transaksjoner. Til tross for potensialet til ISCP i energimarkedene, evaluerer få eksisterende studier ytelsen. De fleste ytelsesvurderinger er basert på den forrige versjonen av IOTA og er dermed ikke relevante og utforsker ikke mulighetene til ISCP. oppgaven presenterer en av de første vurderingene av skalerbarheten til ISCP i forhold til energimarkedene.

Ved hjelp av en designvitenskapelig metodikk ble det gjennomført tre iterative designsykluser, noe som resulterte i opprettelsen og evalueringen av tre artifakter. Disse utnytter smarte kontrakter for å støtte tre nøkkelfunksjoner i energimarkedet: informasjonslagring, handelsmatching og kontraktverifisering. Ved å designe og teste disse artifaktene bidrar oppgaven til utviklingen av skalerbare DLT-baserte løsninger for energihandel i desentraliserte energisystemer.

Resultatene av denne studien indikerer at standardnodekonfigurasjonen var i stand til å håndtere 30 bud per sekund, og at deling av nettverket i mindre undernett er avgjørende for å opprettholde ytelsen og skalerbarheten til DLT-energimarkedene. Dette er i tråd med den nåværende hastigheten av Ethereum blockchain, men ISCP tilbyr den ekstra fordelen at små lokale energimarkeder kan operere samtidig og forankre seg i den globale ledgeren. Med den implementerte matchingsalgoritmen var det maksimale antallet bud per auksjonsrunde mellom 480 og 540. Avhengig av variabler som antall noder i nettverket og maskinvaren nodene opererte på, varierte den totale utførelsestiden for å bekrefte en transaksjon fra 4,56 til 15,1 sekunder. Selv om dette funnet er i tråd med andre studier relatert til DLT og kravene til energimarkedene, faller det under ytelsen som tilbys av andre lagløsninger.

Denne studien legger grunnlaget for fremtidige undersøkelser av bruken av

ISCP i energimarkedene. Det er behov for ytterligere forskning for å evaluere ytelsen under ulike forhold og undersøke de potensielle fordelene med å implementere DLT i sentraliserte energimarkeder, og utforske muligheten for et desentralisert estimeringssystem med flere noder for økt motstandskraft.

Preface

This thesis is the final part of my Masters of Science degree in computer science at the Norwegian University of Science and Technology (NTNU) in Trondheim. I would like to thank Professor John Krogstie at the Department of Computer Science, my supervisor for this project, for his support and guidance which was very much appreciated. I would also like to thank Gleb Sizov at Aneo and +CityxChange for suggesting and supporting this project for my thesis.

Contents

Figures

Tables

Code Listings

Acronyms

DAG Directed Acyclic Graph. 1, 8

DER Distributed Energy Resources. 1

DLT Distributed Ledger Technology. 1, 2

DSO Distribution System Operator. 4, 5, 26

ISCP IOTA Smart Contract Protocol. 1, 2, 33

LEM Local Energy Market. 2

LFM Local Flexibility Market. 26–29

PoW Proof of Work. 9

TSO Transmission System Operator. 4, 5

Chapter 1

Introduction

Many small-scale Distributed Energy Resources (DER) in the market can affect the current distribution networks and their dependence on weather conditions can introduce challenges to the power system [1]. Although DERs are closer to the point of energy consumption and can therefore reduce the strain on the grid, the fluctuations of most DERs, such as photovoltaic (PV) panels and wind power, can cause load balancing issues. To overcome these challenges, various technologies can be used to better monitor and manage the grid in real time. The Smart Grid provides a means to connect and utilize these DERs, thus creating an integrated energy system capable of managing the supply and demand of electricity in an efficient manner. These technologies range from smart meters and sensors that collect and transmit data to advanced analytic tools that can help predict power outages and minimize losses. Solutions such as distributed energy storage systems, demand response programs and battery technologies can help achieve more efficient electricity generation and delivery.

Traditional centralized energy systems currently dominate energy markets, but the increase in distributed energy sources motivates research into distributed control systems. Distributed Ledger Technology (DLT) is an approach to storing and sharing data among multiple data storages. Some main benefits of DLTs are trustlessness, openness, reliability, untamperability and traceability [2]. The technology allows transactions and data to be recorded and shared between a distributed network of participants, thus reducing dependence on a centralized entity to control the system. However, the scalability of the DLT and especially blockchain is an escalating issue when the usage of DLT increases. Bitcoin can only process 7 transactions per second (TPS) [3], while Ethereum can process between 30 and 40 TPS [4]. IOTA is a DLT which instead of the blockchain utilizes a Directed Acyclic Graph (DAG). The IOTA foundation introduced the IOTA Smart Contract Protocol (ISCP) in 2021. ISCP is a framework that extends the base protocol of the IOTA ledger by introducing a multichain environment where all chains are anchored on the IOTA Ledger. Each chain, called a wasp chain, is a separate ledger with smart contracts, functionally equivalent to Ethereum smart contracts. Theoretically, the ISC can scale up to hundreds of thousands of smart contract transactions per second [5].

The +CityxChange[1] project aims to develop a framework and tools to create a common energy market. The services include a whole range of architecture levels from assets in a physical infrastructure to technology and data storage solutions. +CityxChange is a European Union-funded project that aims to develop and test new solutions for energy-efficient and sustainable cities. The project is being led by the city of Trondheim in Norway and involves a number of partners from across Europe. This thesis is in collaboration with Aneo (previously TrønderEnergi), which is a part of the +CityxChange project. To enable running an energy market on DLTs there is a need for storing consumption and production information, placing bids and offers, generating contracts between buyers and sellers, and comparing the contracts to the actual amounts consumed and produced. Creating a decentralized platform will allow energy to be exchanged in real time between all energy market stakeholders. In this study, the technical feasibility of the ISCP is evaluated and compared to the current state-of-the-art solutions related to blockchain and the possible roadblocks.

1.1 Goals and research questions

This thesis aims to present the required knowledge and current solutions to ensure scalability for using DLT in electricity markets. The paper aims to develop a Local Energy Market (LEM) based on the IOTA tangle and the newly released IOTA smart contract Protocol. A private IOTA network is set up to develop the energy market and a wasp chain is started. Afterwards, a smart contract is created through design and creation methodology, which enables the submission of bids and offers, as well as the generation and verification of contracts based on logged consumption and production. The results are evaluated, tested and compared with existing literature. The advantages and disadvantages of the ISCP are discussed, and limitations and challenges are explored. Through this study, the following research questions are answered:

- **RQ2** Is using IOTA Smart Contract Protocol (ISCP) a feasible solution?
    - **RQ1.1** Is storing data in the ledger possible within a suitable time frame and storage possibility?
    - **RQ1.2** Can auctions be performed on the ledger?
    - **RQ1.3** Can verification of contracts be performed on the ledger?
- **RQ2** Are there other roadblocks to the implementation of DLT?

1.2 Contributions

This thesis aims to support the ongoing global green energy transition by deploying software for operation and participation in local energy and flexibility markets.

[1]https://cityxchange.eu/

It also adds to the limited number of studies exploring the IOTA smart contract protocol in general, and specifically in the energy markets. In this way it contributes to researchers and developers in this field. All of the performance tests are based on the previous version of IOTA which is outdated and does not consider the evaluation of IOTA Smart contract functionality. Furthermore, these evaluations mainly target the performance of the DAG and it is necessary to evaluate the IOTA Smart Contract Protocol.

1.3 Thesis structure

Chapter 1 introduces the thesis and its motivation.

Chapter 2 presents relevant theory to understand the findings and results in the subsequent chapters. The definition and core concepts of energy markets and DLTs are explained, in addition to relevant studies from the previous literature review.

Chapter 3 presents the research methodology used and its relevance to the thesis.

Chapter 4 clarifies the architecture of the +CityxChange project and describes the prototype development.

Chapter 5 presents the results of the research.

Chapter 6 contains a discussion of each research question in relation to the results from Chapter 5. The results are also compared with findings from other research.

Chapter 7 concludes the research questions and lists future work.

Chapter 2

Background Theory

Section 2.5.1, 2.5.2 and 2.6. of this chapter is based on a previous structured literature review conducted in spring 2022 [6]. While the content remains largely unchanged, some structural and grammatical changes have been made for the purpose of this thesis.

2.1 Smart grid

The definition of the smart grid includes a variety of different systems. Typically, it is a system consisting of numerous components, subsystems, and functions that are controlled by a control system. In addition to a bidirectional flow of information, the smart grid features a bidirectional flow of energy to establish a distributed energy delivery system and enable the power grid to respond dynamically to events. A traditional power grid transfers power from a small number of producers to a large number of consumers, whereas a smart grid allows participants to use tools to monitor and manage their own energy usage and even sell excess energy back to the electrical grid. A prosumer is a term used to describe a person or a company that both produces and consumes electricity. However, the increase in renewable energy sources installed by end-users can have an impact on the grid, as system operators do not have complete control over them. The Internet of Things (IoT) has become increasingly popular in recent years, particularly in the context of smart grids [7]. In this context, IoT devices can help to manage the grid better by providing real-time data and analytics to help them make informed decisions. By providing real-time data and analytics, IoT devices can also help reduce wastage and promote the use of renewable energy sources.

2.2 Power trading today

TSO & DSO

Distribution System Operator (DSO) and Transmission System Operator (TSO) are companies that play critical roles in ensuring reliable and efficient operation of an electricity grid. In Norway it is Statnett, a state-owned company, which is the TSO responsible for the transmission of electricity over long distances. On the other hand, Elvia is an example of a Distribution System Operator (DSO) in Norway. and is responsible for the distribution of electricity to end-users within specific geographic areas. The main difference between TSO and DSO is the scope of their responsibilities. DSO focuses on distributing electricity within a specific geographic area while TSO focuses on transmitting electricity from generation sources to distribution networks. The position in the network is depicted in Figure 2.1. Both DSOs and TSOs are responsible for maintaining and upgrading of their respective networks. They also manage the supply and demand of electricity and oversee the connection of new customers or generation sources to the network. They work together to ensure that electricity is delivered to customers safely and efficiently.

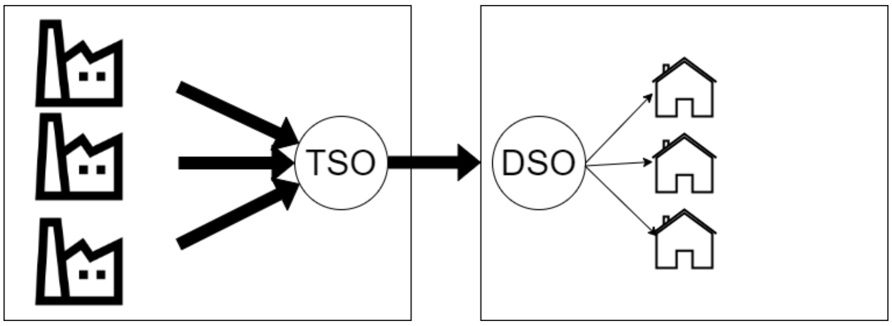


Figure 2.1: A simplified representation of the TSO/DSO structure [6]

Energy trading

Electricity markets that rely on day-ahead and intraday trading are designed to help balance supply and demand for electricity in real time. Nordpool[1] is a company that operates the Nordic power market and is responsible for the day-ahead and real-time trading of electricity in Norway among other things. It also facilitates the trading of electricity between generators and consumers.

The day-ahead market is a market where electricity is purchased and sold for the next day in time-specific blocks, and it is based on factors such as weather, prices, maintenance, and forecasted demand and generation. This allows electricity generators to plan their production and consumers to plan their energy use more efficiently. In this market, prices are determined based on supply and demand predictions. The generator and consumer can take advantage of the day-ahead market to lock in prices, in this way reducing the uncertainty of price volatility.

The intraday market is a spot market where electricity is bought and sold on the same day, and it is based on the actual demand and generation. Nordpool

[1]https://www.nordpoolgroup.com/

offers trading in 15-minute, 30-minute and hourly time blocks to meet the needs of different market areas. This market is used to balance the supply and demand of electricity in real time and to respond to unexpected changes from day-ahead trading.

2.3 Demand response and flexibility

Demand response and flexibility are related concepts that refer to ways in which the electrical grid can be managed to balance supply and demand for electricity. Demand response refers to programs and mechanisms that incentivize customers to reduce or shift their electricity usage during times of high demand. The goal of demand response is to help utilities and grid operators manage demand so that the operators can avoid or reduce the need for additional power plants and ensure that the grid is not overloaded. The concept of demand response was introduced in 1987 [8], but its relevance continues to grow due to the nature of the new energy sources such as photovoltaic (PV) panels.

Flexibility, on the other hand, refers to the ability of the grid to adjust to changes in supply and demand in real time. This can include both changes in supply, such as fluctuations in wind or solar power generation, and changes in demand, such as variations in electricity usage by customers. The goal of flexibility is to ensure that the grid remains reliable and resilient, even when faced with uncertainty or disturbances [9]. To sum up, demand response is focused on shifting or reducing customers' usage during peak hours, whereas flexibility is the overall ability of the grid to adjust to the changing demand and supply.

2.3.1 Local Energy markets

Local energy markets (LEMs) refer to small-scale, localized electricity markets typically operated by a distribution system operator (DSO) or other local entity. These markets allow local energy resources, such as solar panels, wind turbines, and energy storage systems, to participate in the electricity market and sell their excess energy to other customers in the same area. They are also typically more localized and focused on serving a specific geographic area, such as a neighborhood or community. Local Energy Markets (LEMs) provide a platform for local energy resources so that they can participate in the electricity market. LEMs can be divided into three main categories [10]. **Peer-to-peer markets** allow direct trading between individuals and businesses without an intermediary agent, which enables selling and buying energy from neighbours. **Community self-consumption** appears when co-located energy prosumers trade their surplus energy in a market, which allows for more efficient use of energy resources and can help to reduce the need for expensive transmission and distribution infrastructure, by reducing the reliance of the grid. Finally, **Transactive energy (TE)** is a term used for systems that enable machine-to-machine communication and exchange energy between coordinated participants. It is often associated with large-scale implementations which aim to improve the grid and energy availability for larger areas. Transactive energy is a more advanced concept that combines the use of technologies such as smart grid and distributed ledgers to enable real-time management of the electrical grid.

2.4 Distributed ledger

A distributed ledger (DLT) is a consensus-based system in which duplicated and synced data are distributed among a network of peers. In this system new information is added to the ledger; it is then digitally signed, and afterwards replicated across all other nodes. The immutability of data, a core principle of DLT, is maintained through various forms of data structures which will be explained in Section 2.7 Data Structure.

In a distributed ledger the network of peers, also known as nodes, plays a critical role in maintaining the integrity and security of the ledger. Each node is a computer that is connected to the network and stores a copy of the ledger. The nodes work together to validate and process new transactions. At the same time they are also in charge of keeping the ledger in sync. Each node, or participant, is identified by a unique address called a public key that is generated through the use of a private key. A private key is a secret code that is used to sign and authorize transactions, while a public key is a code that is used to identify the node or participant.

Bitcoin was the first cryptocurrency that used a ledger [11]. It was created in 2009 by an individual or a group using the pseudonym Satoshi Nakamoto. Although Bitcoin originally intended to disrupt financial markets, the decentralised principles of DLT have since been applied to a wide range of fields and have gained significant interest. Secondly, Ethereum is a decentralized, open-source blockchain platform that enables the creation of smart contracts and decentralized applications (dApps) [12]. Ethereum is currently the most widely used smart contract platform which is built on DLT. Finally, IOTA is an open-source platform that aims to enable secure, efficient and scalable transactions between devices on the Internet-of-Things (IoT) network, and also support the creation of new IoT-based applications and services. IOTA uses a different approach to the traditional blockchain technology used by most cryptocurrencies. Instead of using a chain of blocks to record transactions, IOTA uses a directed acyclic graph (DAG) called the Tangle which is a data structure that allows for fast and fee-free transactions [13].

2.5 Data Structure In Distributed Ledgers

Blockchain

At its core a blockchain consists of a series of blocks containing a list of transactions. The transactions are grouped together in a block, and each block is linked to the previous block with a cryptographic hash, as seen in Figure 2.2. This creates a chain of blocks, which is built on top of each other and thus contains a record of all transactions which have occurred on the network up to that point.

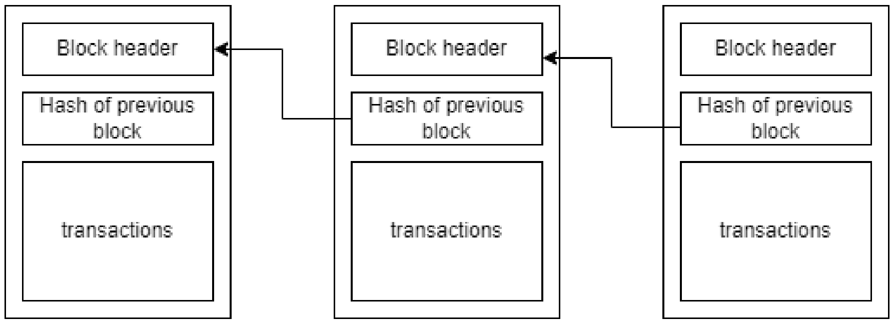


Figure 2.2: Blockchain data structure

DAG

Compared to Bitcoin and Ethereum, IOTA builds on a Directed Acyclic Graph (DAG) which they call ”The Tangle”. They argue that a typical DL consists of two distinct types of participants: those who issue transactions and those who approve transactions. The main idea of the tangle is that if a user issues a transaction, they must also approve other transactions. The general idea is that by approving a transaction, one also indirectly approves all of the predecessors' transactions. DAG is a data structure where each transaction refers to a number of previous transactions. A comparison with blockchain can be seen in Figure 2.3. One of the disadvantages of blockchain is the sequential nature of the blocks, where all nodes must reach a consensus before releasing a new block which can waste resources. Instead of being limited to one block, the DAG offers several blocks to which new blocks can be attached. In this way transactions are processed in parallel instead of in sequence[14].

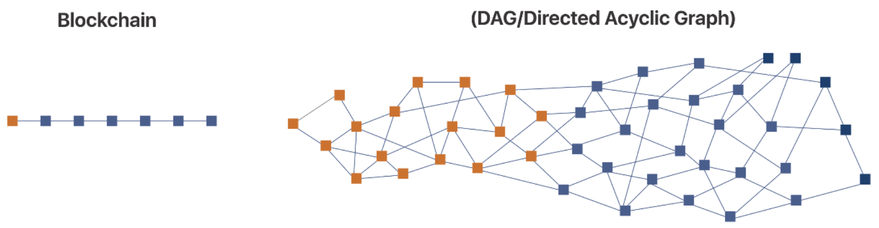


Figure 2.3: Blockchain vs DAG. From Central Blockchain Council of America [15]

2.5.1 Consensus Algorithms

This section is based on a previous structured literature review conducted in spring 2022 [6]. While the content remains largely unchanged, some structural and gram-

matical changes have been made for the purpose of this thesis. In addition was the section ***Adaptive Proof of Work*** *added.*

The use of consensus algorithms helps to prevent the occurrence of malicious nodes in the network. The DL is a distributed network and may not have a centralized authority that can verify the blocks. As a consequence, the network has to work efficiently, even with dishonest nodes. This is called the Byzantine Generals problem [16]. The most common consensus algorithms are Proof of Work, Proof of Stake, and Proof of Authority.

Proof of Work

Proof of Work (PoW) utilizes the workload as a safeguard to prevent malicious nodes, which may create a disincentive for malicious actors because of the computational cost. The proof of work algorithm for Bitcoin uses the previous block's hash in addition to the new transactions to generate a hash. The block can be accepted when this hash is smaller than the threshold. This is essentially bruteforcing a hash function until a solution is found. [17] estimated the task to use approximately 45.8 TWh per year.

Adaptive Proof of Work

IOTA does not have miner fees, and it is, therefore, necessary to add a rate control mechanism to not exceed the allowed throughput based on the available resources. The mechanism should stop spam attacks and malicious attacks, but not limit the rate for honest nodes. The main idea is to count the number of messages recently issued by a node and increase the difficulty for that node to reduce the number of messages it is able to send with the same amount of computational power[18].

Proof of Stake

Proof of stake was developed as an energy-saving alternative to Proof of work. Proof of stake uses cryptocurrencies as an alternative safeguard instead of computational power. Initially, Ethereum utilized the Proof of Work (PoW) consensus algorithm to reach consensus across its network, but it has now transitioned to a new algorithm known as Proof of Stake (PoS). This decreases computational power and energy usage by 99.98% [19]. Instead of malicious agents having to ”invest” in computational power, they have to invest in the cryptocurrency and stake it. A penalty can be deducted from their cryptocurrency if an invalid block is proposed. The distribution of cryptocurrencies defines the frequency with which a node is allowed to generate new blocks. For example, if a user has 5% of the cryptocurrencies in the blockchain, there is a 5% chance of being the next block creator.

Proof of Authority

The Proof of Authority algorithm restricts the addition of blocks to the blockchain to predetermined nodes. In the initial block of the blockchain (genesis block) authorized nodes are specified and it is their responsibility to add subsequent blocks. Since only trusted nodes can add blocks, there is no need for a computational safeguard against malicious nodes. It is more energy-efficient than proof-of-work since the identities of the nodes are utilized as a deterrent, otherwise the identity of the malicious node would be exposed.

2.5.2 Main types of distributed ledgers

This section is based on a previous structured literature review conducted in spring 2022 [6]. While the content remains largely unchanged, some structural and grammatical changes have been made for the purpose of this thesis.

Public Ledger

A public or permissionless ledger allows any user to join the ledger network without restrictions on the actions available. This is the most decentralized type of ledger and the most transparent because anyone can access prior transactions. Public ledgers are often associated with digital currency whose value gives incentives for participants to behave fairly and offset the costs of participating in the consensus algorithm. Due to their decentralized nature, they are often associated with proof of work and are a reason for criticism because of their high energy consumption.

Private Ledger

A private or permissioned ledger is an invite-only network operated by a group of participants. The central authority determines who can join the network where all nodes are known and trusted. The main advantage of a private ledger is that it allows private data to be shared between trusted entities.

Consortium Ledger

A consortium ledger is a hybrid of private and public ledger. The access is invite-only and everyone can add transactions. However, block validation is performed by a group of pre-selected nodes and is only valid if multiple nodes verify and sign the block. Since only pre-selected nodes are able to add blocks, there is an inherent centralization. As fewer nodes participate in the consensus, a consortium ledger is susceptible to attacks on central institutions.

2.5.3 Smart contracts

Smart contracts can be defined as computer protocols that facilitate, verify, and enforce contracts between two or more parties [20]. They also mediate and monitor transactions and enforce contractual clauses. Ethereum is currently the most widely used smart contract platform. When a smart contract is executed, data is added, modified or removed from the distributed ledger. The smart contract is added to the ledger by generating a transaction with the compiled code. This secures the code and makes it impossible to tamper with. Once the smart contract is deployed, it cannot be updated.

In the EVM any transaction that changes the state in the ledger has an associated cost, paid in gas. Gas in Ethereum refers to the computational units used to execute smart contract operations on the Ethereum Virtual Machine (EVM). It acts as a measurement of the computational effort required to execute a specific operation and serves as a fee paid by the contract invoker to incentivize miners to include the transaction in the blockchain. Gas is an important mechanism in Ethereum that helps maintain the network's integrity by mitigating the risk of denial-of-service (DoS) attacks and ensuring that network resources are allocated fairly [21]. Every transaction must specify the quantity of gas willing to consume. Each operation has a predefined cost, including database reads and writes, and every computational step has a pre-defined cost, and the total cost of a transaction is: Cost of transaction = gas limit * gas price.

View Call

A "view call" is a synchronous invocation of a smart contract used to retrieve information from the ledger without modifying the state of the smart contract [5]. Since they do not modify the state, they do not require a transaction to be posted. The view is run in the context of the current state and returns the requested information to the caller of the view.

Request Call

A "request call" is a way to invoke a smart contract function [5]. The request can modify the smart contract state and perform actions that would change the ledger state. Requests are signed by the private key of the caller. An important note is that if a view is called within a request call, the view requires gas to run.

Ethereum

Ethereum utilizes the Ethereum Virtual Machine (EVM) to execute the transactions. The Ethereum Virtual Machine (EVM) is a decentralized, virtual machine that executes smart contracts on the Ethereum network. These contracts are stored on the blockchain and the EVM executes them when the conditions specified in the contract are met. One of the key features of the EVM is its Turing completeness which means that it can perform any computation that can be expressed in a finite number of steps [20]. This allows developers to build various applications on the Ethereum platform, including financial transactions, supply chain management, and more. In addition to its ability to execute smart contracts, the EVM also includes features such as a memory store, a stack for storing data, and a set of opcodes (operations codes) for performing various functions. These features enable the EVM to perform complex computations and execute smart contracts. The smart contracts are written in high-level coding language, such as Solidity, and compiled into source code.

IOTA

IOTA smart contract Protocol (ISCP) works as a Layer 2 (L2) extension on the IOTA Ledger. [18] The concept of "Layer 2" refers to solutions built on top of a base protocol, often called "Layer 1". Each wasp chain is run by its own set of validator nodes. To produce a valid threshhold signature a quorum of $\lfloor 2N/3+1 \rfloor$ validators is required to validate a set of transactions[5]. Each validator node owns a private key which produces a partial signature. These validators agree on the new state of the ledger and settle them on the layer 1 protocol, thereby making the state immutable and irreversible. Layer 2 solutions take transactions off-chain and only occasionally settle them on-chain, as seen in Figure 2.4.

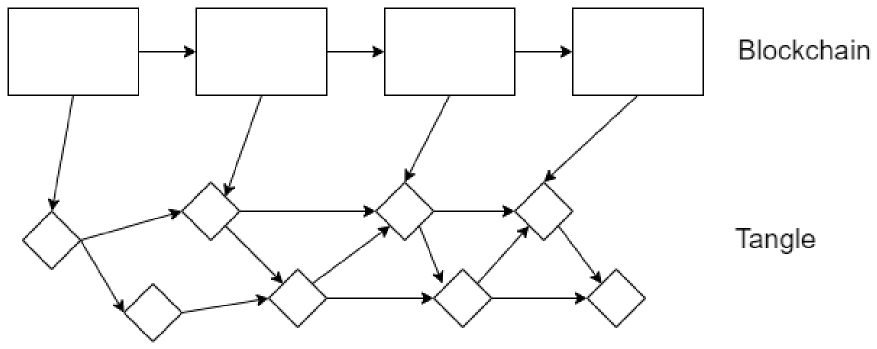


Figure 2.4: Blockchain data structure

Each L2 chain has its own state and smart contracts, and that is why the Layer 1 has the ability to run multiple blockchains in parallel. Each group of validators can control their own Layer 2 blockchain and therefore increase throughput. Each wasp chain consists of one or more wasp nodes as depicted in Figure 2.5.

The Virtual Machine (VM) is a part of the chain responsible for state transitions on the ledger. The VM includes core smart contracts and built-in interpreters such as WebAssebmly (wasm) interpreters and EVM interpreters. It supports EVM/Solidity smart contracts and brings the development from years of implementations on Ethereum.

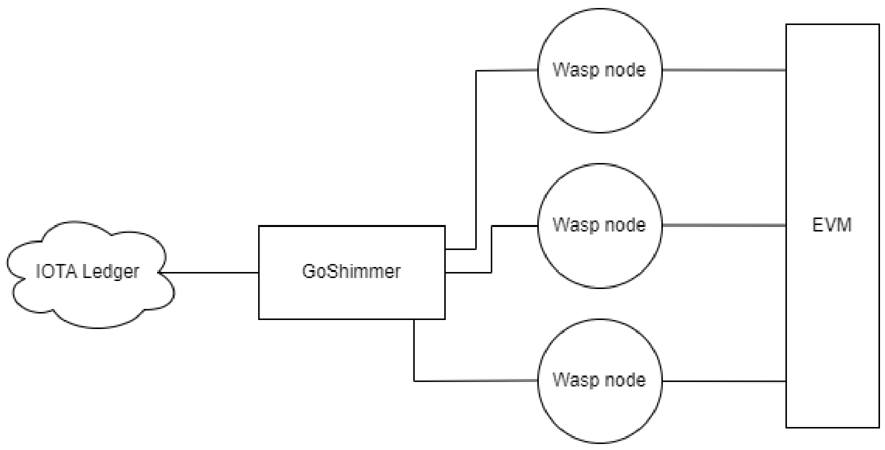


Figure 2.5: Network architecture *ADD JSON-RPC*

The communication with the smart contract is performed through Remote Procedure Call (RPC) API. This enables the Python class to interact with a network node, and to read and write to the ledger. Using the web3.py package requires setting up the environment to connect to the correct wasp chain and the correct smart contract.

2.6 Previous Structured Literature Review

As mentioned previously, this section is based on a previous structured literature review conducted in spring 2022 [6]. While the content remains somewhat unchanged, this provides a summary of the findings and some structural and grammatical changes have been made for the purpose of this thesis.

Spring 2022 was devoted to gaining knowledge on a topic of choice related to the master's degree in computer science at NTNU. The research goal and research questions were as follows:

- **RQ1** *Acquire an overview of the field of blockchain for demand response*
- **RQ2** What are the current solutions for the use of blockchain in demand response?
  - **RQ2.1** What are the current solutions to ensure scalability?
  - **RQ2.2** What are the current solutions to ensure privacy?
  - **RQ2.3** What are the current solutions to ensure security?

2.6.1 Previous Structured Literature Review

The SLR was conducted in the spring of 2022 as a preparation project whose goal was to acquire knowledge about the research topic. The research fields for the SLR were the emerging interest in the DLT and its relation to energy markets.

The purpose of conducting a structured literature review was to learn about the domain of interest. A more systematic approach has the benefit of reducing biases in the research. Planning consists of identifying the need for the SLR, creating the research questions, and developing and reviewing the protocol for the later processes. The search was performed in Google Scholar, which used research databases such as ScienceDirect and IEEEXplore.

The selected query was:

```
"distributed ledger" AND "demand response" AND
"flexibility" AND "prototype" AND "smart grid"
```

The literature study aimed to find a more manageable number of studies. Based on the selected search phrase, 177 related publications were found. Initially, the exclusion criteria (EC1-5) found in table 2.1 were used to reduce the number of publications. The time frame of the papers ranged mainly from 2018 to 2022, and only 17 were published before 2018. Articles which appeared multiple times in the search due to publicising them in different journals were removed together with the papers that were not accessible. Using the exclusion criteria, the number of papers was reduced to 119. The method to identify if the papers met the primary inclusion criteria IC1, IC2 and IC3 was to read the titles and abstracts. The criteria are found in Table 2.1.

The secondary inclusion criteria IC5, IC6, and IC7 ensure that the papers are related to scalability, security, and privacy implementations. The process of controlling if the paper passes the secondary criteria was skim-reading through the papers. IC3 was also used for the secondary process because after skim-reading, it was found that some papers initially included did not have any implementation or prototype, which greatly reduced the number of papers that satisfied the criteria.

Criteria Identification	Criteria
EC1	The study done before 2018
EC2	The study not written in English
EC3	Paper without empirical evidence
EC4	The paper is not accessible or available
EC5	The study is not a duplicate
IC1	The study is in the field of distributed ledger
IC2	The study focuses on demand response
IC3	The study presents a prototype or implementation
IC4	The study ensures scalability
IC5	The study ensures security
IC6	The study ensures privacy

Table 2.1: Exclusion Criteria

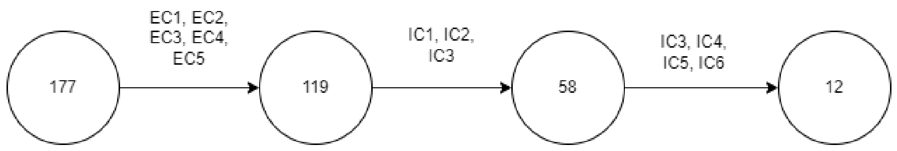


Figure 2.6: Selection process

2.6.2 Results

The previous SLR resulted in findings related to scalability, privacy and security. A significant portion of the literature concentrated on the scalability of the DL. 10 studies were related to scalability out of 12 studies found in the preliminary study. Second-layer solutions, aggregation of data and sharding are some of the possible tools to utilize to ensure scalability. In addition, some papers compare the cost associated with running on ledger technologies. The distribution of papers can be seen in Figure 2.7. A selected subset of these studies were chosen to be included in the background of this thesis.

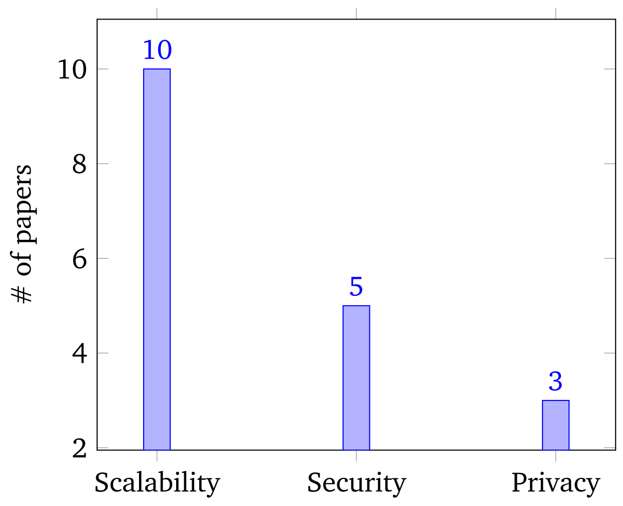


Figure 2.7: Distribution of papers related to topic.

2.6.3 Scalability

By reducing the amount of data that needs to be submitted to the blockchain, including more transactions in a block becomes possible, ultimately reducing transaction delay. This approach involves sending a digital fingerprint to the blockchain and using the immutability of the blockchain to verify the database. Implementing a second-layer database can significantly increase throughput by reducing the number of transactions submitted to the distributed ledger. However, the downside of a database layer is that it requires the maintenance of two systems.

Another approach to handling large amounts of data is implementing an aggregation solution. This approach prevents the transmission of large amounts of data to the blockchain and subsequently to all other nodes, because not all nodes require information about every consumption. Dividing the networks into independent layers may have the benefit of reducing the dependence on a network connection. By enabling trading in a local network, one may utilize some of the demand response even if parts of the network are inaccessible. Aggregation may remove verifiability and transparency, making it more difficult to trace transactions back.

C. Pop et al. [22] improved scalability by proposing a scalable second-tier approach that combines the blockchain and a NoSQL database to register data to the blockchain less frequently while leveraging the scalability of NoSQL for off-chain energy data. Real-time energy data is maintained outside of the blockchain in their proposed system, and every 30 minutes, a single transaction including all of the energy data is submitted to the blockchain. Each aggregated energy transaction is monitored and validated against the strategy of the smart contract, and prosumers that deviate from the original plan may receive a penalty. Their off-chain configuration handled up to 50,000 transactions per second.

Following the same approach, C. Pop et al. [23] collects energy data and stores the real-time data in a scalable, off-chain database. A deviation can be determined by comparing real-time data with the flexibility request. This is recorded on the blockchain and serves as a fingerprint for the off-chain data associated with the transaction. As only the deviation is transacted, the blockchain is freed from maintaining the real-time data. In addition, they claim that each prosumer may be required to send only one transaction per hour, which would significantly limit the number of transactions.

A. Lucas et al. [24] present a proof of concept for a demand response registry on the HyperLedger Fabric to evaluate its viability and performance. They tested the concept with 3, 10, and 28 participants and demonstrated that the total execution time was unaffected by the number of participants. However, execution time increased exponentially if the number of transactions per second exceeded 32. In addition, they discovered that the number of participants and transactions increased the probability of errors. The total execution time for 28 nodes and 64 transactions was 12 seconds. The implementation was a laboratory size, with 1 DR provider, one aggregator, and one battery storage system.

2.6.4 Cost

M. Foti and M. Vavalis [25] published at the time the first large-scale simulation which compared a decentralized blockchain-based market with a centralized implementation. They simulated up to 1000 houses and 40 producers and compared three scenarios with different levels of decentralization. The goal was to find the different costs based on how much was done on the DLT and if it was economically feasible. They created a blockchain network consisting of 250 nodes, where 40 of these were generators. They argued that since the power grid already relies on regulators, there is some inherent centralization, and it is therefore suited to use a private Ethereum consortium blockchain. As they wanted a specific clearing time, they used a PoA consensus algorithm. The first scenario has all the computation done on the blockchain, and one particular account was responsible for clearing the market. This resulted in a cost of $11.80 for each user per day. The first scenario, where the computations are on-chain, provided another challenge. The node responsible for calculating the market-clearing had an associated cost of $4304 per day. The second one was a random account that cleared the market, which resulted in $15.85 per day. In the third approach, the blockchain was only used for a consensus on the transactions, while the market-clearing was computed externally, and the cost associated was $4 per day. The cost reported was with a clearing time of 15 minutes because a more frequent clearing time resulted in higher costs. The third approach, where the blockchain is only used as a reference and there is little to no computation on the blockchain was argued to be the most secure.

2.6.5 Sharding

CEnTra is an application of sharding in blockchain proposed by S. Mitra et al. [26]. They develop a hierarchical model and extend the original sharding presented by ChainSpace. Due to the fixed location of consumers and producers, the sharding can consider their physical location when they are mapped to a shard because it is more likely to transfer power to nearby nodes. A transaction containing inputs from different shards has to lock the objects in their shard and proceed with the execution. By preventing inter-shard communication, as few as possible objects are locked, and a minimal amount of shard has to execute the code. One node is selected to represent the entire location within an area. When an area has an excess or deficit of energy, the representative node can transact with other locations. The hierarchical model enables the propagation of energy transactions between locations, which can continue at higher levels. This allows it to propagate information and energy differences to the different agents in a city-scale implementation, such as consumers, transformers, sub-stations, distribution networks, and transmission networks. In this way, the nodes can transact without being aware of the energy availability of other locations. The basic random sharding in ChainSpace increased the throughput by 59.52% compared to no sharding. Compared to the random sharding in ChainSpace, their shard mapping increased the performance by 39.57% for 250 inputs.

2.6.6 Consensus Algorithm

Since the inaugural Bitcoin whitepaper, the consensus algorithm has developed and continues to grow, [27]. Consensus algorithms have a significant impact on the throughput of the blockchain. Multiple papers claim that the PoW consensus algorithm is not efficient enough to guarantee scalability. As stated in Table 2.2, only one of ten articles concerning scalability utilized the PoW consensus algorithm. The majority of articles utilized a private blockchain, which can boost throughput by restricting who can join the network and minimizing the processing requirements to secure the blockchain. The previous literature review resulted in studies for multiple ledger technologies. The technologies, consensus algorithms and whether they were public or private are shown in Table 2.2

Report	**Platform**	**blockchain**	**Consensus**
[26]	ChainSpace	private	Other
[28]	Ethereum	private	PoW
[22]	Ethereum	Private	PoA
[25]	Ethereum	Private	PoA
[24]	HLF	Private	Other
[23]	Ethereum	Private	PoA
[29]	QuarckChain	Unknown	Other
[30]	HLF	private	Other
[31]	HLF	Private	Other
[32]	Ethereum	Public	PoA
[33]	HLF	private	Other

Table 2.2: Different consensus algorithms used in the related reports. HLF: HyperLedger Fabric

A. Mandal [34] found in their study that some of the consensus algorithms, which are in use, do not protect privacy. They implemented improvements to the algorithms to prevent consumption/production data from being stored directly on the blockchain, where everyone can read it. The goal of anonymization or pseudonymization is to make smart meter data unlinkable to its originator. In the paper, they found that there are some cases where it is possible to link the smart meters to their users by comparing the pseudonym attributes with the user's attributes. They also found that total anonymization is not suited for electricity providers because they need to be able to invoice the users.

2.7 Verifiability

In addition to the proposed Electron Volt Exchange (EVE) architecture for scalability by S. Saha et al. [30], they also provide a method for verifying the measured consumption and production. They introduce a new notion of distributed measurement verification to validate and cross-validate the power injected into the system and the measured power from the sensor measurements. They focus on the power flow between aggregators and verify the prosumer's transacted energy vs. the actual energy.

The second-tier solution presented by C. Pop et al. [22] also ensures that data has not been tampered with. They present a tamper-evident registration of smart meter data using digital fingerprinting on the off-chain sensor data to be linked to the on-chain transactions. They altered the input energy data and saw that their solution could identify tampering. Afterwards, they were able to estimate the real input value. This resulted in less than a 0.75% difference from the monitored data. Due to the temper-evident feature, the impact on the energy forecasting process was less than 5%.

The transparency of the blockchain is often a desired feature and makes it possible to audit and bring openness. Energy data is private information and should not be stored on the blockchain using the prosumer's digital identity. GDPR, which aims to protect the personal information of users, also institutes laws one must adhere to.

In a standard blockchain implementation, the prosumers' monitored energy data is published to the blockchain. Even though there is no link between the prosumers' address and their identity, a malicious agent can establish this relation, which is why it is called a pseudo-anonym system. C. Pop et al. [23] solve this by proposing a zero-knowledge proof to prevent the exposure of private information in the blockchain. On top of the blockchain technology, they define a decentralized demand response implementation. Zero-knowledge proof guarantees the privacy of prosumers' energy data. The system provides a demand request that users attempt to fulfil. The users report the resulting demand response and if the result is validated they can receive financial compensation.

Chapter 3

Research Method

3.1 Design Science Methodology

Design science is important because it provides a systematic and rigorous approach to solving complex problems and research questions. The Design principles followed in this research were influenced by the work of [35] and [36]. Their design principles involve a structured process of identifying problems, defining objectives, developing and evaluating potential solutions, and finally communicating the results. This process helps to ensure that the resulting artifacts are effective, useful, and relevant to the needs of the stakeholders in a specific context. The context may include people, businesses, organizations, and existing technologies that are relevant to the problem. The Design science methodology also helps to identify any potential limitations or challenges that may need to be considered when developing an artifact. The guidelines presented by Peffers et al. in [35] are as follows and consist of 6 steps which are described below and shown in Figure 3.1.

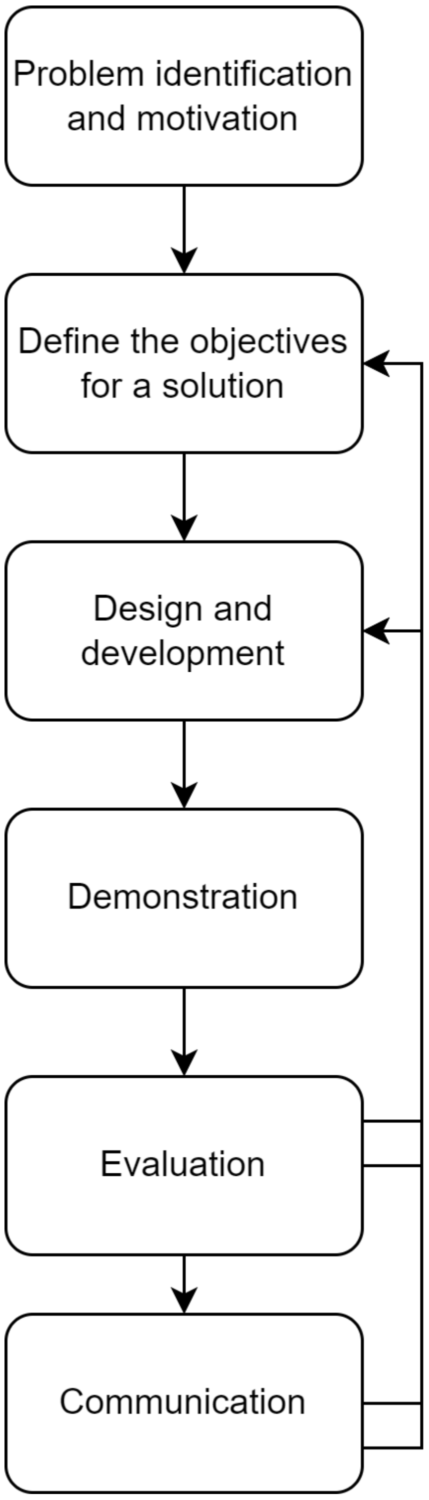

Problem identification and motivation
Define the objectives for a solution
Design and development
Demonstration
Evaluation
Communication

Figure 3.1: design principle process

3.1.1 Activity 1: Problem Identification and Motivation

The first step was to identify the problem which needs to be solved. This research was performed in connection with the digital economy NTNU project[1] which tries to transform the digital economy, thus allowing for changing scale and scope of economic activities, transactions, products and services. In addition, NTNU is part of the CityxChange project which aims to change city blocks to be energy positive by finding solutions to utilize tools such as flexibility. The previous literature review and discussions with the CityxChange project were important to identify the task and gain motivation. The decision landed on utilizing the newly implemented ISCP. However, the previous literature review did not result in studies related to the ISCP or IOTA in general. It was therefore necessary to perform a new literature review.

Refresh of state-of-the-art

The previous SLR did not yield any IOTA-related findings. Therefore, a second SLR was necessary for finding the prior work related to IOTA. Due to the limited number of studies related to IOTA Smart contracts, the query used for finding articles was greatly expanded. Instead of solely focusing on energy markets, the related studies included all implementations of IOTA smart contracts. The smart contract protocol on IOTA is relatively new, with very few papers published related to the topic. To focus on the smart contract implementation on IOTA, the query "WASP" was added. Since this is a relatively new term, this greatly reduced the number of related studies. The query used to perform the SLR was:

```
"IOTA" and "WASP" and "Smart Contract"
```

12 studies were found in the search. Since there was a very limited number of studies, all the studies could be fully read through and there was no need for reducing the number of studies based on title and abstract. The inclusion criteria presented in Table 3.1 were used to eliminate studies. IC1 and IC2 ensured that the study presented an implementation of the new ISCP. Then IC3 and IC4 ensured that the study was either related to energy markets or performance.

IC Identification	Criteria
IC1	The study focuses on IOTA Smart Contract Protocol
IC2	The study presents a prototype or implementation
IC3	The study focuses on Energy markets
IC4	The study focuses on performance

Table 3.1: Inclusion Criteria

[1]https://www.ntnu.edu/digital-transformation/digeco

Results

The following two papers were selected from the literature review as they are relevant to the topic under discussion.

C. Mullaney et al. [37] demonstrated that an IOTA wasp network was advantageous for users and enhanced transfers between prosumers. Each agent in the microgrid was equipped with a smart meter which was responsible for forecasting future energy consumption. Their network consisted of three wasp chains that could trade with each other. Each microgrid was composed of five Photovoltaic agents and was connected to its own Wasp chain and could interface directly with the smart contracts without affecting other microgrids. This enabled them to utilize the layer 2 frameworks that the wasp chains offered. Consequently, the microgrid could balance its supply and demand before querying other microgrids. Communication with other microgrids could be done through the Layer 1 ledger. They employed a double auction with a uniform price as the market mechanism. They demonstrated that IOTA and ISCP used far less energy than conventional blockchains and also showed an average of 166.41% benefit for the microgrids and a 6.3% benefit for the consumers. The results were obtained by utilizing the smart contract off-ledger. They ran the smart contracts off-ledger to assess the benefit of the market mechanism and utilized a python script and real-world data.

J. Rosenberger et al. [38] detailed results from two industrial use cases and experiments were presented on a private DLT network based on IOTA in the Industrial Internet of Things (IIoT), focusing on the resource demands for IIoT devices with different network setups. Two use cases were presented, where the second use case required Smart contract functionality. The results confirmed the suitability of IOTA for IIoT devices. Furthermore, an overview of the required resources of the IIoT devices with different transaction rates and network sizes was provided. An average response delay of 24 seconds and an average CPU usage of 20The results confirm the suitability of IOTA for IIoT devices. Furthermore, an overview of the required resources of the IIoT devices with different transaction rates and networks sizes are given. They achieved an average response delay of 24 seconds and an average CPU usage of 20%.

3.1.2 Activity 2: Define the objectives for a solution

It is important first to identify the problem the artifact is intended to address. The next step is to identify the wanted outcomes , such as improving efficiency, reducing costs, or addressing a particular challenge. Identifying specific metrics that can be utilized to evaluate the performance might also be beneficial. Activity 1 identified the problems and laid the foundation for generating the objectives of this paper. Research question 1, **”Is using IOTA Smart Contract Protocol (ISCP) a feasible solution?”**, was the main objective of the study. The limited number of studies present a gap in the cross-section between energy markets and the IOTA smart Contract Protocol. Through the discussion with Aneo, it was decided that three possible solutions could be further looked into. This laid the basis for **RQ1.1**

Is storing data in the ledger possible within a suitable time frame and storage possibility? , **RQ1.2 Can auctions be performed on the ledger?** and **RQ1.3 Can verification of contracts be performed on the ledger?**

3.1.3 Activity 3: Design and development

The core of design and research is the development of a new artifact that deals with the research problems or objectives. This might involve designing and prototyping of a new artifact. The artifact should be designed to perform a specific function, and be feasible in the given timeframe with the available resources. It should also provide utility for the intended users or stakeholders and have a meaningful contribution to the research field.

Initially, it was planned to utilize the native Rust and Go implementation to create smart contracts. However, quickly some roadblocks appeared early in the implementation and it was decided to utilize the EVM support and the corresponding Solidity implementation.

It was decided to split Research question 1 into three artifacts. These parts build on top of each other and make it possible to test different components of the energy market separately in relation to different use cases. These questions are also affected by the methods used for testing and evaluating the prototype. The design and development of the artifacts are described in Chapter 4.

3.1.4 Activity 4: Demonstration

A demonstration is an important part of design science because it validates the usefulness and effectiveness of the artifact. It is achieved through empirical testing and evaluation, which can include interviews and experiments. The demonstration may be in a controlled laboratory setting or in a real-world setting to assess its performance and impact. The design and development were supported by two types of data generation and observations.

The smart contract will be tested in a simulated environment to ensure that it behaves correctly under various conditions. This involves creating a variety of test scenarios to simulate different conditions. The contract will then be executed in the simulated environment, and the results will be observed to ensure that it behaves as expected. This is done continuously while developing the contract.

3.1.5 Activity 5: Evaluation

The evaluation focuses on assessing the effectiveness and utility of the artifact that was developed in the previous steps. It might involve conducting experiments or case studies to test the artifact under different conditions, or collecting feedback from users or experts in the field. It should be considered if it is feasible to use the artifact in a real-world setting, including factors such as cost and ease of use.

After passing simulated testing, the smart contract will be deployed to a test net for further testing. This involves going from unit testing to a more real-life scenario and running the system on multiple node count inputs to test scalability. The first research question aims to determine the technical limitations of the solution. To evaluate this, the focus will be on five aspects of the solution: total time used to confirm a transaction, the size of the ledger, total time used to retrieve information from the ledger, CPU usage, and gas usage. These metrics provide a comprehensive picture of the efficiency and scalability and help understand the limitations and strengths of this DL platform. The success of the artifact will be measured against the performance of other implementations on different DLTs and in addition measured against the requirements of energy markets. The data extracted from the testing is quantitative, and the previous research referred to in Chapter 2.6 makes it possible to extrapolate qualitative insight based on quantitative data.

The python module ”time”[2] has been used to measure the transaction times. To measure the CPU and memory usage, the ”psrecord”[3] was used, a tool based on the psutil[4] library, which can retrieve information on running processes and system utilization (CPU, memory, disks, network, sensors) in Python.

The second Research question **”Are there other roadblocks to the implementation of DLT?”** is related to the limitations related to the approach and problems related to other areas. This question requires an exploration of the different aspects of the implementation. Since design science is inherently an iteration-based approach, different aspects of the solution will be explored, and limitations will also be discovered during the development.

3.1.6 Activity 6: Communication

Effective communication is essential because it helps ensure that the results are disseminated and can be used by others to solve similar problems. This study aims to meticulously follow all the necessary steps and provide valuable insights to the scientific community. Chapters 4., 5., 6. present both the technical aspects of the artifact as well as its implications.

[2]https://docs.python.org/3/library/time.html
[3]https://github.com/astrofrog/psrecord
[4]https://github.com/giampaolo/psutil/

Chapter 4

Prototype development

This section will describe how the system was implemented. Firstly, it will discuss the architecture of the +CityxChange Local Flexibility Market, before it goes into details of the implementation. The prototype development was divided into three iterations, where each iteration created an artifact which was tested. The first artifact allows a user to securely log all the necessary information needed to enable the DLT implementation. The second artifact is used to ensure the functionality of the market maker who can facilitate trades. The third artifact is designed to verify and validate that the correct amount of resources has been consumed or produced based on the agreement.

4.1 +CityxChange architecture

+CityxChange is a project trying to change city blocks and districts from being energy negative to energy-positive [39]. This process includes finding solutions to utilize tools such as flexibility and other renewable resources. The project's approach and solutions for establishing local Positive Energy Blocks (PEBs) are more than just obtaining a balance between local energy consumption and production, but rather optimising available and viable local renewable energy sources in order to scale local PEBs to the district level. A scalable and efficient PEB is dependent on systems and solutions being able to utilise this flexibility in order to obtain a balance between local production/utilisation of renewables and local energy consumption.

The traditional end-customer can be an active market participant by providing both locally generated power and utilising available flexibility in consumption. When prosumers generate energy today, they must traditionally sell it to the DSO since it requires information about the electricity delivered into the grid to maintain grid quality. As the first organization globally, +CityxChange was granted a special permit to sell energy locally [1]. These are the participants of the Local Flexibility Market (LFM) [40]:

[1]https://www.tu.no/artikler/forst-i-verden-med-kjop-og-salg-av-strom-i-nabolaget/518324

- **Assets and asset owner:** Energy resources which consume, and/or produce and store energy in the market. These are often referred to as consumers, producers or prosumers.
- **Asset integrator:** Integration services, ABBOPTIMAX ®, that provide communication between the market operator and the asset management system.
- **Trading platform:** Digital service used for trading power between assets in LFM which in the case of +CityxChange is Volue.
- **Market operator:** Services to support LFM operations, in this case Aneo.

The architecture and communication between the actors in the Local Flexibility market is shown in Figure 4.1

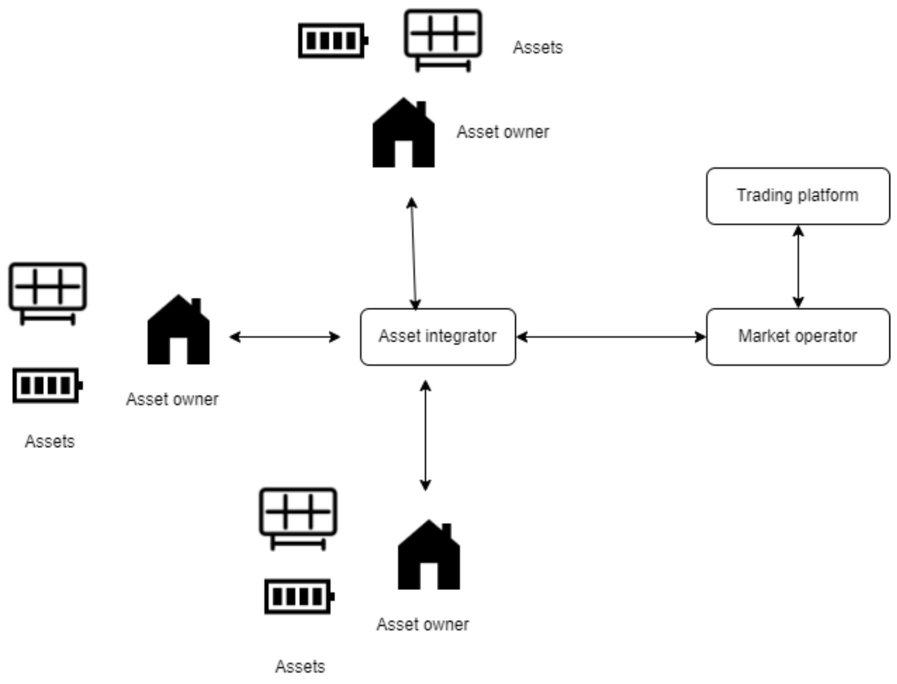


Figure 4.1: Overview of architecture

+CityxChange utilizes three market rounds and the timeframe of the different rounds is presented in Figure 4.2.

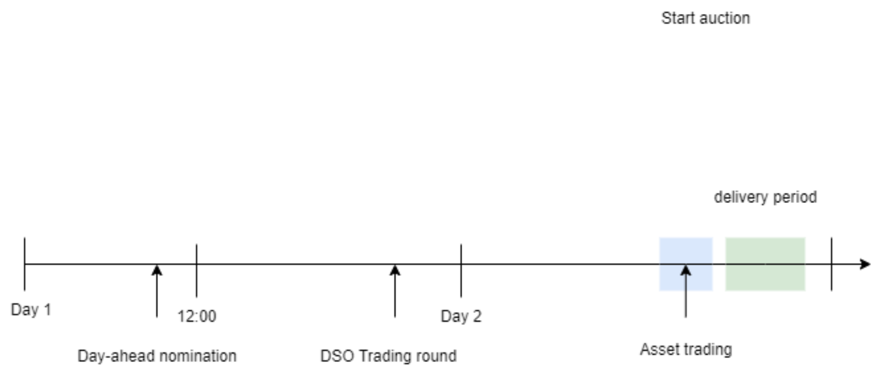


Figure 4.2: Timeline of different market rounds

Day-ahead

The market operator will submit the predicted net load for all assets to the market platform (Nordpool). There is no energy trading in this round, but the nominations represent the best estimate of the load profile for the following day. Nominations consist of energy usage per hour and are submitted for the following day. This is a part of the balancing obligation for the market operators as they integrate LFM into the global market.

DSO trading

The day-ahead trading round forecasts the energy balance for the following day. There is always a chance that the real energy balance may differ from this prediction. As a result, +CityxChange allows the DSO to purchase flexibility from the assets in the LFM. Keeping energy use below a certain threshold can be helped by lowering the amount of energy needed.

Asset trading

The asset trading round means trading between assets in a specific trading window for the following timeslot or the nearest future. This gives the assets a greater awareness of the present and anticipated production and consumption. The market operator currently generates bidding curves for each asset for the next delivery period and the trades can be used to transmit control signals to the asset integrator to turn on or off certain components.

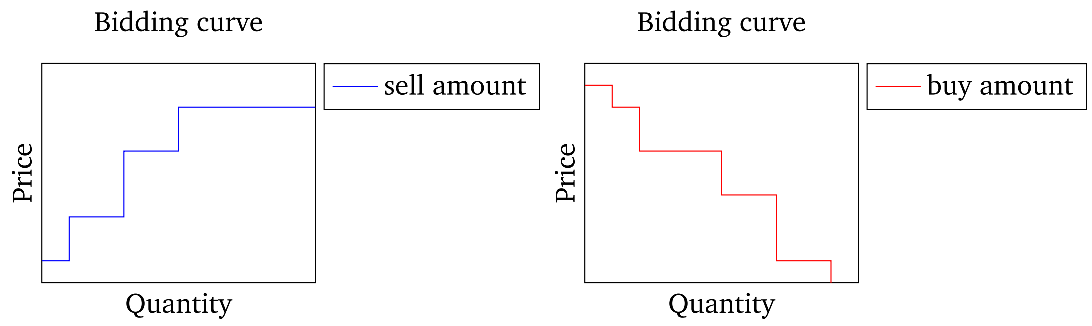


Figure 4.3: Bidding curves for buy and sell

4.2 Specifications and architecture

An Local Flexibility Market (LFM) is a large and complex system with many actors and processes. Initially, the IOTA ledger was planned to be used only as a verification method to make the transactions immutable. Through the literature review and discussions with Aneo and after examining the complexity of the LFM, it became apparent that IOTA could be used to a larger extent than just a verification method. When IOTA was utilized, the platform could be extended to enable users to store production and consumption information, place or match bids and offers, as well as verify contracts. The proposed solution focuses on the ”asset trading” round in the +CityxChange architecture. The interface between market operator and its trading platform was used to create the required functions and seamlessly integrate them into the current architecture.

The interface consists of the following functions illustrated in Figure 4.4:

- Sending bids to the platform
- Seeing all bids in the auction
- Starting and running the auction and creating the contracts
- Returning all contracts

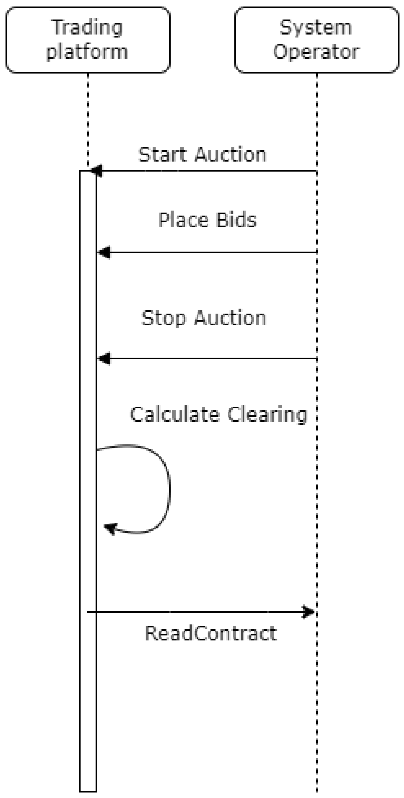


Figure 4.4: Current sequence diagram

As depicted in Figure 4.5 the DLT implementation can be viewed as a standalone solution that is capable of communicating with multiple providers in the system. There are two alternative bidding methods. The first one, shown as the dotted line **Market operator - DLT** in Figure 4.5, refers to the centralized initialized transactions with which the market operator engages the smart contract on behalf of the users. This permits the delegation of particular tasks to the system operator while maintaining the immutability and trustlessness of the smart contract and DLT. Additionally, this connection is needed for the Market Operator to verify and settle the contracts.

The second one, a user-initiated transaction, permits the user to govern the participation and execution of the smart contract in a decentralized manner. This is shown as the dotted line **Asset integrator - DLT** in Figure 4.5. This connection is also needed for the asset integrator to retrieve the contracts and the subsequent usage for the next time period. Each alternative has its own advantages and disadvantages. The decentralized transaction encourages decentralization and user autonomy, whereas the centralized transaction can enhance efficiency and scalability. The choice of method will depend on the specific use case and the acceptable trade-offs. Currently, the bids and offers are generated from the system operator, but since the goal of DLT is to be decentralized, both methods will be investigated. The market operator generates and submits the bidding curves to the trading platform for each asset for the subsequent delivery period.

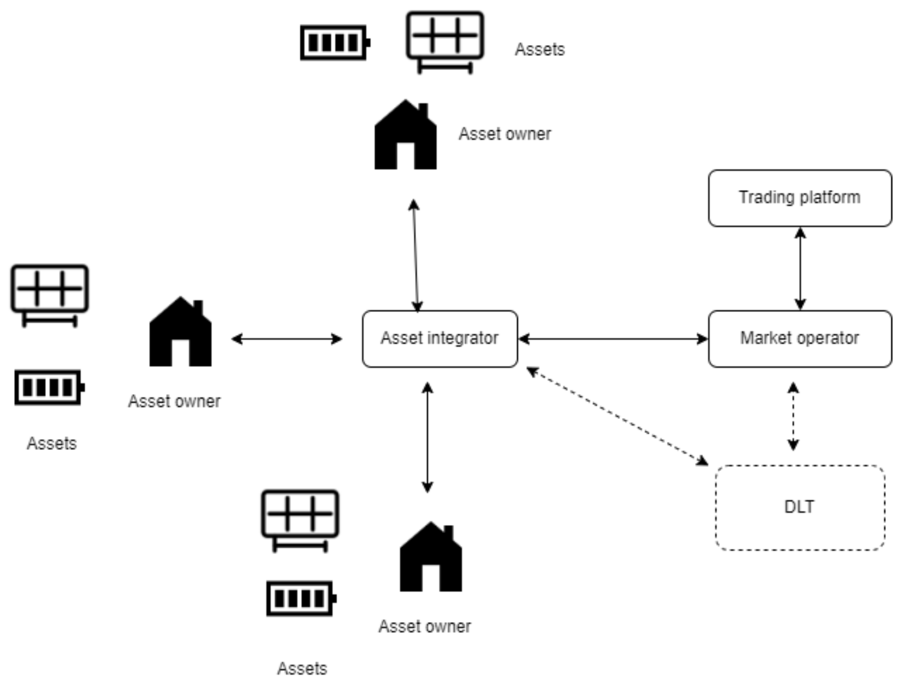


Figure 4.5: Overview of architecture

4.2.1 Functional requirements

To ensure the successful development of the prototype, it is essential to establish a clear set of functional requirements. These requirements outline the essential features and capabilities that the prototype must possess to meet its intended goals. Additionally, they serve as a reference point throughout the development process, helping to ensure to remain focused on building a solution that meets the following requirements:

1. Log electricity consumption and production
2. Create a new auction round
3. Place a bid or offer
4. Generate contracts based on the bids and offers
5. Verify consumption and production compared to the contract

The requirements reflect the existing interface from the market operator, and criteria 1. and 5. were added to enable verification, all of which are necessary to create a Local flexibility market. The market operator does not have a directly compatible interface for logging energy consumption and production to the ledger, but this is added to the proposed solution to make sure it delivers the functions needed to achieve the local flexibility market. The technical requirements and the resulting sequence of actions are depicted in Figure 4.6.

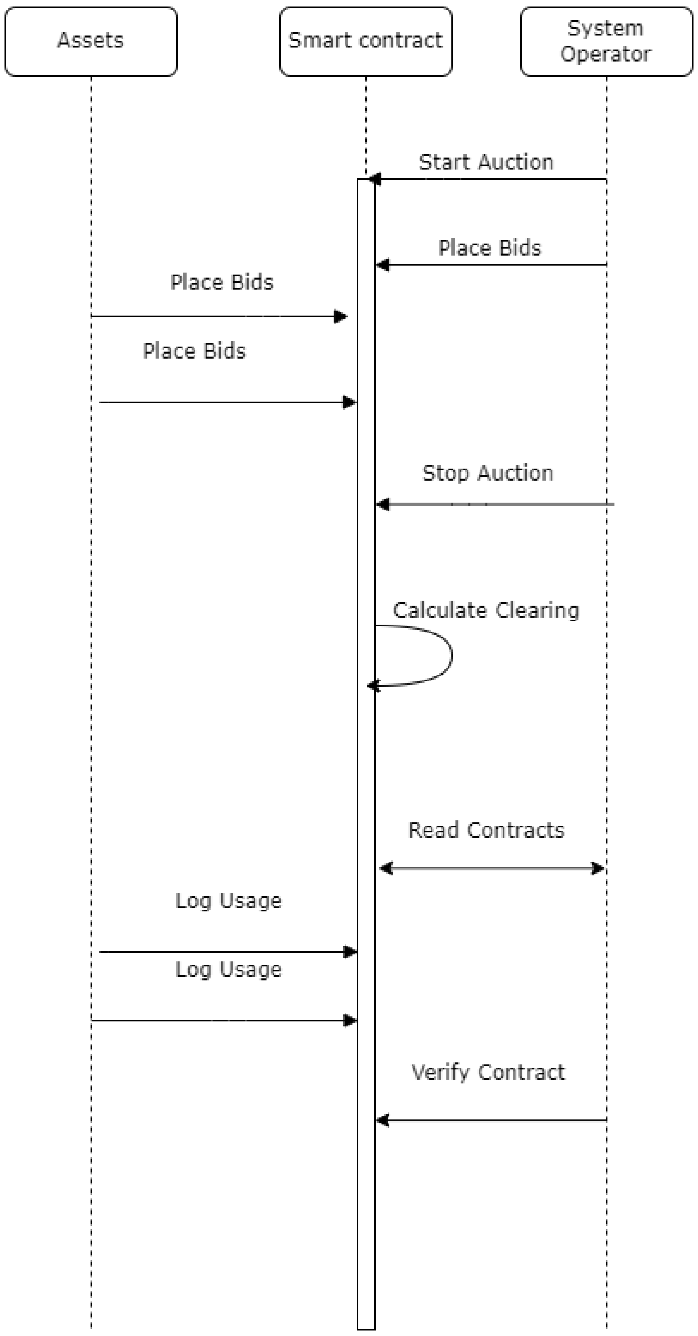


Figure 4.6: Proposed sequence diagram

4.3 Development tools and technology stack

4.3.1 Solidity

Rust, Go, and Solidity are programming languages supported by the ISCP. After investigating the potential of each programming language, Solidity was selected. Solidity is a high-level, object-oriented programming language designed for smart contracts. It is intended for the Ethereum Virtual Machine (EVM), and it draws inspiration from C++, Python and Javascript. Solidity's advantage is the number of resources and documentation generated by years of Ethereum development. Numerous Solidity tools and tutorials simplify the development process for ISCP. By employing it for ISCP, developers may take advantage of the same capabilities that have made Ethereum smart contracts so successful. It also allows the implementation to be tested on EVM-compatible DLTs.

4.3.2 Remix

Remix IDE is a no-setup tool for developing smart contracts. It facilitates smart contract testing, debugging and deployment. In addition, it permits testing and debugging on a local instance of the ledger. As a result, Remix IDE provides a useful and convenient method for developing smart contracts, enabling developers to rapidly deploy and test smart contracts prior to committing the contract to an actual distributed ledger.

4.3.3 JSON-RPC

An application needs to be connected to a node to be able to read or write data to or from the ledger. For this purpose, the wasp node exposes a JSON-RPC which defines data structures and rules, and transfers the data over an HTTP socket. This enables easier interaction between the network and is used by many libraries.

4.3.4 web3.py

Web3.py makes interacting with blockchain networks easy and convenient for Python developers, allowing them to use the same powerful language they already know to access Ethereum and other DLT protocols. It is an implementation of web3.js for Python, allowing users to access Ethereum and other DLT networks using Python code easily. It offers an API to interact with remote DLT nodes over JSON RPC. [2].

4.3.5 Metamask

Metamask is a browser extension that in combination with Remix enables fast deployment and interaction with Smart Contracts. Metamask enables users to con-

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

nect their browsers to DLT networks and provides an easy-to-use interface for transactions, signing messages, and deploying smart contracts. Metamask makes it easy to access and interact with decentralized applications (dApps), creating a much more user-friendly experience than manually managing wallet keys and connecting to the DLT network.

4.3.6 Jupyter Notebook

Jupyter Notebook is a tool that allows users to create notebooks and documents that can contain live code, equations, visualizations, and narrative text. It is widely used for data science and machine learning and has quickly become a valuable tool for DLT developers who can take advantage of the same capabilities to develop dApps. By using Jupyter Notebook, developers can interactively write and execute code, analyze data, embed visualizations and share documents in an easy-to-use interface.

4.4 Setup

Setting up the IOTA ledger[3] and the WASP chain [4] can be a challenging effort. It is recommended to follow the guidelines provided by the IOTA Foundation. A quick overview of the steps is as follows:

1. Start the Docker image, which initialises the IOTA ledger
2. Initialize a wasp client and connect it to the IOTA network generated in Step 1, or the public network if requested
3. Start the Wasp node or nodes with the correct configuration
4. Deploy the wasp chain
5. Request funds from the faucet
6. Deposit funds from the L1 to L2 for transactions
7. Deploy the smart contract
    a. Save the smart contract address and the ABI for the smart contract
    b. Utilize the smart contract address and ABI to communicate with the smart contract

To generate a wasp chain with multiple nodes, each node has to trust each other. This task is done by distributing the PubKey and NetID from each node to all other nodes which should be in the network.

```
wasp-cli peering info

PubKey: xx
NetID:  127.0.0.1:4000
```

[3]https://wiki.iota.org/shimmer/smart-contracts/guide/development_tools/docker_preconfigured/
[4]https://wiki.iota.org/shimmer/smart-contracts/guide/chains_and_nodes/installing-wasp/

4.4.1 Interact with the Wasp chain

To interact with the wasp chain requires the web3.py package. Each wasp node exposes a JSON-RPC to be able to communicate with the ledger. To be able to call a smart contract, the chain address, smart contract address and ABI are required. Here is an example of these:

```python
import web3

self.w3 = web3.Web3(web3.Web3.HTTPProvider("http://IP:9090/chain/CHAINURL/evm/
    jsonrpc"))
self.abi = '[{"inputs":[{"internalType":"uint256","name":"_buyId","type":"uint256
    "},...'
self.address = *SMART CONTRACT ADRESS*
self.contract_instance = self.w3.eth.contract(address=self.address, abi=self.abi)
self.chain_id = self.w3.eth.chain_id
```

Code listing 4.1: Parameters to interact with smart contracts

The presented code in Code Listing 4.2 is a Python function that enables the creation, signing, and submission of a transaction to a JSON-RPC endpoint, followed by waiting for a reply. This is an example of how to send a transaction, and other transaction types follow the same principles.

The **place_bids** function takes a list of bids as input and performs the following steps. First, it initiates an account using the private key stored in the ”PRIVATEKEY” variable. Then, it constructs a transaction object using the **buildTransaction** method. This transaction is defined with several parameters, including the ”chainId” of the blockchain, the ”from” address of the sender, the ”nonce” value which prevents replay attacks, and the ”gasPrice” value which is the amount of token the sender is willing to pay per unit of gas used to process the transaction. Subsequently, the transaction object is signed using the private key associated with the account. Next, the signed transaction is submitted to the network via the **send_raw_transaction** method, and, lastly, the code waits for the transaction to be processed by the network and a receipt to be generated, using the **wait_for_transaction_receipt** method.

```python
def place_bids(self, bids: List[Bid]):
    myAddress = self.w3.eth.account.from_key(*PRIVATEKEY*)
    call_function = self.contract_instance.functions.placeMultBids(bids).
        buildTransaction({"chainId": self.chain_id, "from": myAddress.address, "
        nonce": self.w3.eth.getTransactionCount(myAddress.address), "gasPrice":
        self.w3.eth.gas_price})
    signed_tx = self.w3.eth.account.sign_transaction(call_function, private_key=
        myAddress.privateKey)
    send_tx = self.w3.eth.send_raw_transaction(signed_tx.rawTransaction)
    tx_receipt = self.w3.eth.wait_for_transaction_receipt(send_tx)
```

Code listing 4.2: Code to generate, sign and send a transaction

4.5 Access Control

Smart contract access control refers to the mechanisms and policies that determine which parties are able to interact with a smart contract and perform certain actions. This is an important aspect of smart contract security as it ensures that only authorized parties are able to execute functions and make changes to the contract.

A Role-Based Access Control (RBAC) was utilized to assign specific roles for users with the appropriate role. When a smart contract is published, the creator is set as its owner. After initialization, one can use the **changeOwner** (Line 13) function to change the owner. Specific functions in the smart contract can only be executed by its owner, providing a secure way to handle operations on the smart contract. This enables functions to have the **onlyOwner** modifier for functions that are intended only to be accessible to the smart contract owner.

A whitelist was added to prevent bids and offers from being submitted without permission. The list can be modified by the owner to prevent bids from unauthorized agents. The owner calls the **addUser** function to add an user to the whitelist. The function to add bids to an auction round uses the **verifyUser** modifier to ensure the address is allowed to place a bid or offer. In order to prevent sub-functions from being called from external users or programs, the **internal/private** modifier can be utilized only to allow the smart contract itself to call the functions.

```
 1
 2      mapping(address => bool) whitelistedAddresses;
 3      address owner;
 4
 5      constructor(){
 6          owner = msg.sender;
 7          whitelistedAddresses[msg.sender] = true;
 8      }
 9      modifier onlyOwner() {
10          require(msg.sender == owner, "Ownable: caller is not the owner");
11          _;
12      }
13      function changeOwner(address newOwner) public onlyOwner {
14          whitelistedAddresses[owner] = false;
15          owner = newOwner;
16          whitelistedAddresses[newOwner] = true;
17
18      }
19      modifier isWhitelisted(address _address) {
20          require(whitelistedAddresses[_address], "You need to be whitelisted");
21          _;
22      }
23
24      function verifyUser(address _whitelistedAddress) public view returns(bool) {
25          bool userIsWhitelisted = whitelistedAddresses[_whitelistedAddress];
26          return userIsWhitelisted;
27      }
28
```

```
    function addUser(address _addressToWhitelist) public onlyOwner {
        whitelistedAddresses[_addressToWhitelist] = true;
    }
```

Code listing 4.3: Functions for access control

4.6 First Iteration

The first iteration was performed to evaluate the possibility of storing information on the ledger. This step focused on creating the structures to enable more complex functionality. The entire smart contract can be found in Appendix A. The first iteration can be utilized independently of the other iterations as a part of a verification process to verify the bids and contracts.

4.6.1 Auction

The **Auction** structure was the base for storing data and executing market mechanisms. When generating a smart contract, if not further specified, the address used to deploy the contract is the owner of the contract. The contract owner is allowed to start a new auction, but if there is already an auction in progress, line 14, **auctions[currentAuction].finished** prevents it. This was to prevent multiple auctions from being run simultaneously. Since only one auction can be run simultaneously, the **currentAuction** specifies the current auction.

```
 1      mapping(uint => Auction) public auctions;
 2
 3      uint public currentAuction = 0 ;
 4
 5      struct Auction{
 6          uint auctionId;
 7          Bid[] bids;
 8          Bid[] offers;
 9          Contract[] contracts;
10          bool finished;
11          uint startTime;
12      }
13      function newAuction(uint _auctionId) public onlyOwner {
14          if (auctions[currentAuction].finished  currentAuction==0)
15              if( currentAuction==0  auctions[_auctionId].auctionId != 0){
16                  Auction storage a = auctions[_auctionId];
17                  a.auctionId = _auctionId;
18                  a.finished =false;
19                  a.startTime = block.timestamp;
20                  currentAuction = _auctionId;
21                  emit auctionStarted(currentAuction);
22              }
23
24          }
25      }
```

Code listing 4.4: Auction structure and newAuction function

4.6.2 Bid

A bid is a buy or sell offer which consists of **volume** and **price**. **userId** is mandatory to link the bid to the user. Since each asset can provide multiple bids, **bidNr** is added. The variable **isSelling**, specifies if it is a buying or selling offer. A user can place a bidding curve consisting of multiple bids. To reduce the number of transactions, a **placeBids** function was added. Solidity does not support floats, and it was decided to multiply each value with 1000, to create numbers with 3 decimal points. This is sufficient in the test scenario.

```
    struct Bid{
        uint userId;
        uint volume;
        uint price;
        bool isSelling;
        uint bidNr;
    }
```

Code listing 4.5: Bid structure

4.6.3 Contract

A Contract is created whenever a seller and buyer are found to be a match for one another. It provides the buyer and seller IDs in addition to the amount and price agreed upon. The addition of contracts is being done to make it possible to verify levels of consumption and production. Through the interface the market operator requires the contract for both the seller and buyer, but to reduce the amount of data stored on the chain, this is generated in the interface, and not stored on the blockchain.

When the selling and buying bids are matched, a contract is generated. The purpose of adding contracts is to enable verification of consumption and production. The contract includes the amount of energy being traded as well as the price per unit which is agreed on.

```
    struct Contract{
        uint buyId;
        uint sellId;
        uint volume;
        uint price;
    }
```

Code listing 4.6: Contract structure

4.7 Second Iteration

The objective of the LFM is fair and equal profit distribution and volume maximization. Profits should not be affected by the order of bids. As a result, LFM seeks to level the playing field so that all bidders have an equal opportunity to compete for the same resources without any bidder or group of bidders being favoured. A double-auction market is a frequently utilized market strategy where suppliers and consumers submit bids for price and quantity. In [10], it was found that 25% of 139 studies implemented a variant of double auction. This finding indicates that the double auction has become a popular choice for energy markets, possibly because of its ability to provide both buyers and sellers with a fair price.

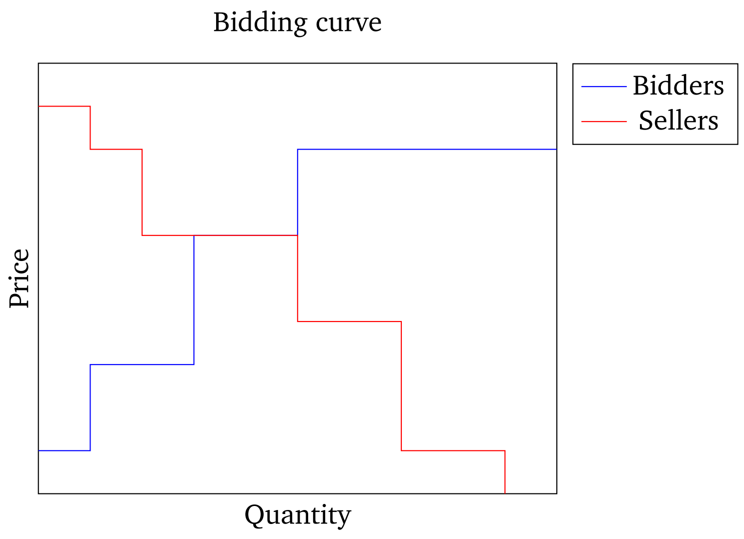


Figure 4.7: Intersection of bids and offers

When a buyer places a bid, they specify the maximum price they are willing to pay for a given amount of electricity. The seller places an offer, specifying the minimum price they are willing to accept for a given amount of electricity. At the end of an auction round, the market operator ends the auction, matches the bids and offers, and finds the clearing price. The bids, b_n, and offers, s_n are sorted based on price as shown below where a quicksort algorithm [41] is used:

$$b_1 \geq b_2 \geq \ldots \geq b_n$$

$$s_1 \leq s_2 \leq \ldots \leq s_n$$

Priority is given to the producer with the lowest unit price and the consumer with the highest unit price by sorting the bids. The lowest and highest prices are matched. The pseudocode for the algorithm is presented in Algorithm 1 and the code can be seen in Appendix A, line 272 to line 398. This trading round continues until either the producers or consumers are depleted, or until all demands have been satisfied. The clearing price is then determined by comparing this price to the bids placed by buyers and offers made by sellers. The current solution establishes a single price for all contracts. This solution is fair and transparent because all parties are charged the same price for the contract, regardless of the amount offered. The solution also prevents unfair advantages and situations in which one party can manipulate the market by submitting an unreasonable bid or offer.

Once the auction is finished, the assets can retrieve the contracts for their respective user IDs and see their commitment in the auction period.

Algorithm 1 Uniform double auction

```
Require: $n \geq 0$
Ensure: $y = x^n$
  $y \leftarrow 1$
  $sortedBids \leftarrow sortDecending(bids)$
  $sortedOffer \leftarrow sortAscending(offers)$
  $bid = sortedBids[0]$
  $offer = sortedOffers[0]$
  $N \leftarrow n$
  while $bid \leftarrow biddersLength, offer \leftarrow offersLength$ do
      if $bid.price \geq seller.price$ then
          if $buy.quantity \geq sell.quantity$ then
              $generateContract(bid, offer)$
              $offer.amount -= bid.amount$
              $getNextOffer(sortedOffers)$
          end if
          if $buy.quantity \leq sell.quantity$ then
              $generateContract(bid, offer)$
              $bid.amount -= offer.amount$
              $getNextBid(sortedBids)$
          end if
      else
          $generateContract(bid, offer)$
          $getNextBid(sortedBids)$
          $getNextOffer(sortedOffers)$
      end if
  end while
  $clearingprice = lastPrice$
```

4.8 Third iteration

The third iteration focused on the implementation of the settlement of contracts. When a contract between bids and offers is made, the subsequent consumption and production data can be used to verify if the contract was fulfilled.

4.8.1 Consumption

Logging of consumption and production is provided in addition to the data related to the auction rounds. Logging is also required to verify that the amounts specified in the contracts are correct.

```
    mapping(uint => Log[]) public logging;
    mapping(uint => mapping(uint => Log)) public logMapping;
    struct Log {
        uint userId;
        int usage;
        bool isSelling;
        uint timeStamp;
    }
```

Code listing 4.7: Consumption and production logging

4.8.2 Verification of the contracts

When designing a system for logging and storing data on a blockchain, it is important to carefully consider the type of data structure to be used. In this case, two different data structures were implemented to store information about energy consumption and production. One of these data structures is a double mapping, which is particularly useful for storing structured information that corresponds to a specific auction round. This type of mapping allows for efficient retrieval of information based on both the auction round and the user ID. On the other hand, an array data structure is more appropriate for logging sporadic events that may not match the auction rounds. This type of data structure is useful when information is logged at irregular intervals.

Once the contracts are created, both parties are anticipated to adhere to the predetermined energy volume, and the amount of energy consumed or produced can be compared against the promised values registered in the contract. A crucial part of the operation is to ensure that market participants receive payment for what they have sold or produced. If the delivered amount does not reflect the agreed amount, the contract is not fulfilled and can be marked as such. This occurs after the predefined market interval is elapsed. For instance, suppose a contract is created between two users for 10 Kwh. In that case, if the seller only provides 5 Kwh, they will receive payment for that amount but must purchase the remaining 5 Kwh from the global market.

To assist with the verification, a starttime and endtime were added to the **Auction** structure, which also enables the auction period to be variable and not based on any hardcoded timeframe. This is why the function **newAuction(uint _auctionId)** was modified to **newAuction(uint _auctionId, uint _startTime, uint _endTime)**

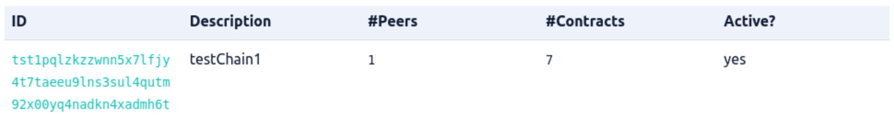
The function **VerifyContract** is called with the auctionId and UserId. The function iterates through the contracts for the auction and checks whether the corresponding logs from the period match the agreed-upon amounts. If there is a deviation, the function returns a flag marking the contract as unfulfilled.

Chapter 5

Results

5.1 Prelimenary tests

By following the procedures outlined in Chapter 4.4, the IOTA ledger and wasp chain setup show that they are operational. The initial step in implementing this system is to activate the hornet nodes, followed by the wasp nodes, and finally deploy the wasp chain. Once the wasp chain has been successfully deployed, it should be available through the Wasp Dashboard. As illustrated in Figure 5.1, the dashboard depicts a deployed wasp chain.



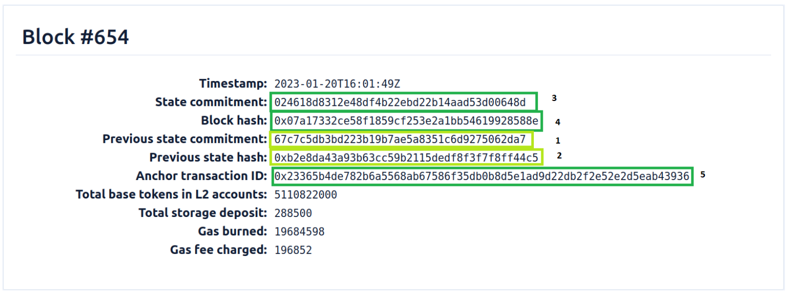
ID	Description	#Peers	#Contracts	Active?
tst1pqlzkzzwnn5x7lfjy4t7taeeu9lns3sul4qutm92x00yq4nadkn4xadmh6t	testChain1	1	7	yes

Figure 5.1: Screenshot from the Wasp dashboard

The testing scenarios were conducted on a private GoShimmer network with up to 4 nodes. The private network was achieved by utilizing the docker private network tool in the ”private_tangle” directory of the Hornet repository [1].

When the network is running, a transaction can be sent through the JSON-RPC to interact with the wasp chain. When a transaction is added to a block, the current block can be explored through the wasp and hornet dashboard. Figure 5.2 shows block #654 which refers to the previous state of the block as seen highlighted as 1 and 2. These two can be found in the tangle presented in Figure 5.3. When the block is committed, the current state and block hash 3,4 can be found in figure 5.4. The hash on the tangle can be seen in the highligted 5.

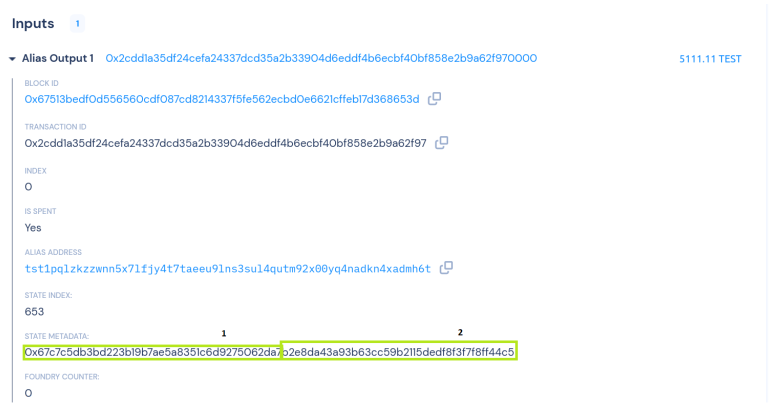
[1]https://github.com/iotaledger/hornet/tree/develop/private$_{t}angle$



Block #654

```
                    Timestamp: 2023-01-20T16:01:49Z
             State commitment: 024618d8312e48df4b22ebd22b14aad53d00648d   3
                   Block hash: 0x07a17332ce58f1859cf253e2a1bb54619928588e   4
    Previous state commitment: 67c7c5db3bd223b19b7ae5a8351c6d9275062da7   1
          Previous state hash: 0xb2e8da43a93b63cc59b2115dedf8f3f7f8ff44c5   2
        Anchor transaction ID: 0x23365b4de782b6a5568ab67586f35db0b8d5e1ad9d22db2f2e52e2d5eab43936   5
Total base tokens in L2 accounts: 5110822000
        Total storage deposit: 288500
                  Gas burned: 19684598
             Gas fee charged: 196852
```

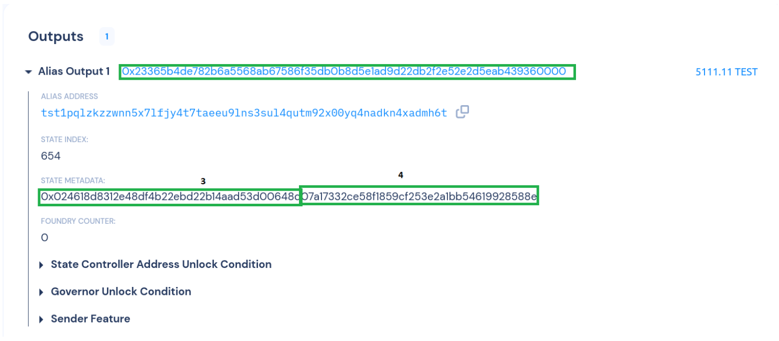
Figure 5.2: Block #654 from the wasp dashboard



```
Inputs 1

Alias Output 1  0x2cdd1a35df24cefa24337dcd35a2b33904d6eddf4b6ecbf40bf858e2b9a62f970000        5111.11 TEST
  BLOCK ID
  0x67513bedf0d556560cdf087cd8214337f5fe562ecbd0e6621cffeb17d368653d
  TRANSACTION ID
  0x2cdd1a35df24cefa24337dcd35a2b33904d6eddf4b6ecbf40bf858e2b9a62f97
  INDEX
  0
  IS SPENT
  Yes
  ALIAS ADDRESS
  tst1pqlzkzzwnn5x7lfjy4t7taeeu9lns3sul4qutm92x00yq4nadkn4xadmh6t
  STATE INDEX:
  653
  STATE METADATA:                          1                                2
  0x67c7c5db3bd223b19b7ae5a8351c6d9275062da7b2e8da43a93b63cc59b2115dedf8f3f7f8ff44c5
  FOUNDRY COUNTER:
  0
```

Figure 5.3: The message added to the tangle seen from the Hornet dashboard



```
Outputs 1

Alias Output 1  0x23365b4de782b6a5568ab67586f35db0b8d5e1ad9d22db2f2e52e2d5eab439360000        5111.11 TEST
  ALIAS ADDRESS
  tst1pqlzkzzwnn5x7lfjy4t7taeeu9lns3sul4qutm92x00yq4nadkn4xadmh6t
  STATE INDEX:
  654
  STATE METADATA:                          3                                4
  0x024618d8312e48df4b22ebd22b14aad53d00648d07a17332ce58f1859cf253e2a1bb54619928588e
  FOUNDRY COUNTER:
  0
  State Controller Address Unlock Condition
  Governor Unlock Condition
  Sender Feature
```

Figure 5.4: The message added to the tangle seen from the Hornet dashboard

5.1.1 Functional requirements

One critical aspect of testing the performance of the prototype is ensuring that it meets the functional requirements that have been established in Chapter 4.2.1. Therefore, it is essential to verify that each requirement has been fulfilled before moving on to testing its performance. If the prototype is not verified to meet each functional requirement, any performance issues may be attributed to missing features or capabilities instead of technical limitations.

The debugging tool in Remix IDE was used to ensure that data structures and functions such as adding bids and offers were correctly generated. This made it easy to test the solution. The first tests of the matching algorithm were performed with the bids presented in Figure 5.5 and Figure 5.6. The contracts were returned as a list of tuples. The figures below give a clearer picture of the connection between the bids and the contracts, where the corresponding bids and contracts are marked in green.

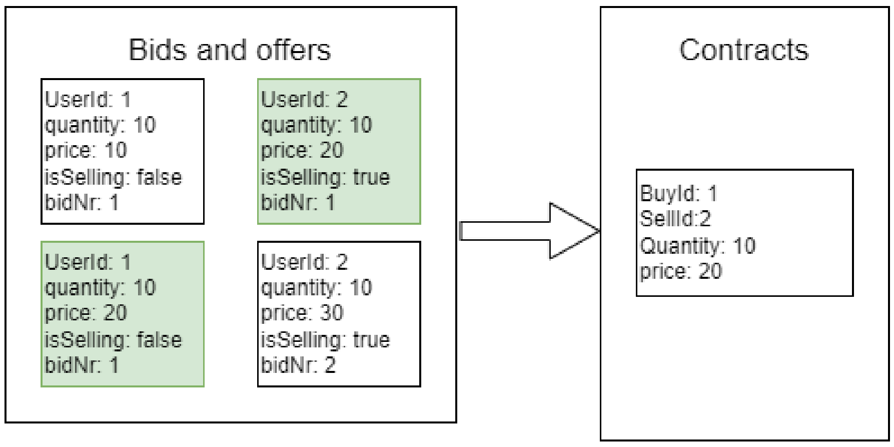


Figure 5.5: First test of contract generation

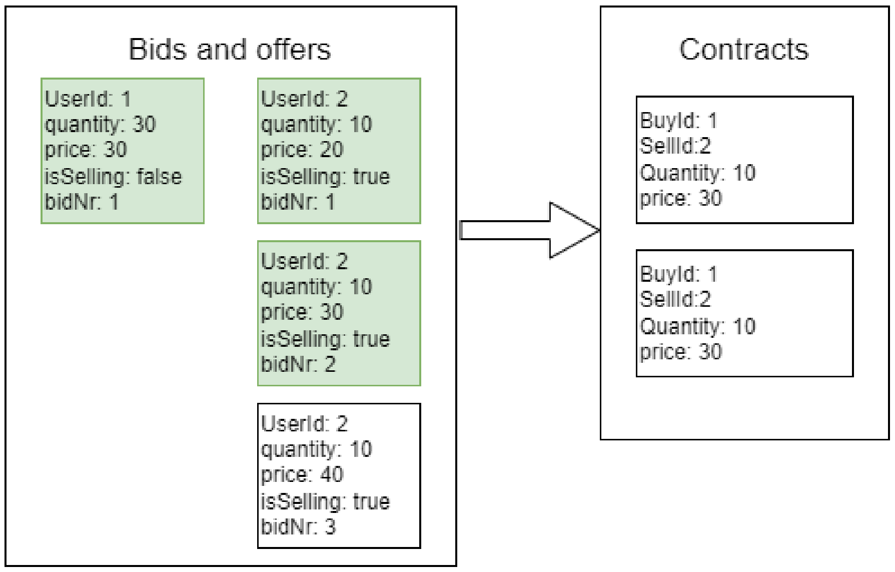
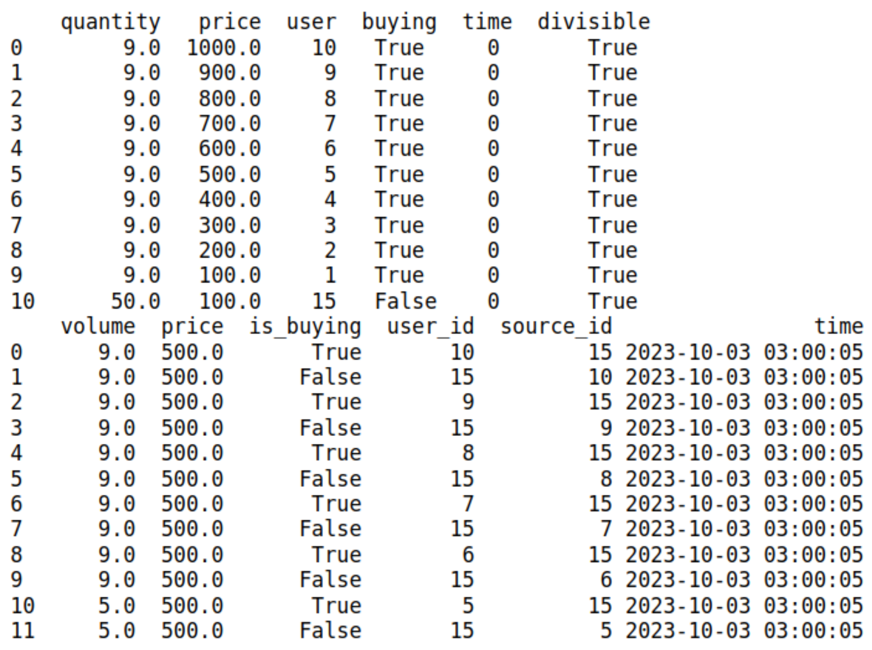


Figure 5.6: Second test of contract generation

The interface for the market operator was also tested at this stage. One of the test scenarios consisted of matching bids and partially fulfilling some orders. In addition, the market operator requires the contract for both the seller and buyer, but to reduce the amount of data stored on the chain, this is generated in the interface, and not stored on the blockchain. Figure 5.7 shows the data presented through the market operator interface. Figure 5.8 shows the result of running the action with 10 buyers and 1 seller. The total demand is 90 but the supply is only 50, which results in the matching of 5 of the bids with the highest price, and a partial fill of 1.

```
   quantity  price  user  buying  time  divisible
0      10.0    2.0     1   False     0       True
1      10.0    2.0     2    True     0       True
   volume  price  is_buying  user_id  source_id                time
0    10.0    2.0       True        1          2 2023-01-03 03:00:05
1    10.0    2.0      False        2          1 2023-01-03 03:00:05
(True, [(1, 2, 1000, 200, 0, 1000)])
```

Figure 5.7: Use +CityxChange interface



```
    quantity   price  user  buying  time  divisible
0        9.0  1000.0    10    True     0       True
1        9.0   900.0     9    True     0       True
2        9.0   800.0     8    True     0       True
3        9.0   700.0     7    True     0       True
4        9.0   600.0     6    True     0       True
5        9.0   500.0     5    True     0       True
6        9.0   400.0     4    True     0       True
7        9.0   300.0     3    True     0       True
8        9.0   200.0     2    True     0       True
9        9.0   100.0     1    True     0       True
10      50.0   100.0    15   False     0       True
    volume  price  is_buying  user_id  source_id                time
0      9.0  500.0       True       10         15 2023-10-03 03:00:05
1      9.0  500.0      False       15         10 2023-10-03 03:00:05
2      9.0  500.0       True        9         15 2023-10-03 03:00:05
3      9.0  500.0      False       15          9 2023-10-03 03:00:05
4      9.0  500.0       True        8         15 2023-10-03 03:00:05
5      9.0  500.0      False       15          8 2023-10-03 03:00:05
6      9.0  500.0       True        7         15 2023-10-03 03:00:05
7      9.0  500.0      False       15          7 2023-10-03 03:00:05
8      9.0  500.0       True        6         15 2023-10-03 03:00:05
9      9.0  500.0      False       15          6 2023-10-03 03:00:05
10     5.0  500.0       True        5         15 2023-10-03 03:00:05
11     5.0  500.0      False       15          5 2023-10-03 03:00:05
```

Figure 5.8: View of bids and offers and the resulting contract through the interface

The VerifyContract function was tested by performing the auction round and logging consumption to the ledger. The first log data did not exceed the amount specified in the contract and returned a **False** flag and the corresponding returned values seen in Table 5.1 on line 1. Afterwards, another logging was performed and the verifyContract was run again. This resulted in a return value of **True** and the corresponding return values seen in Table 5.1 on line 2.

ID	Fufilled	buyer ID	Seller ID	Quantity	Price	Actual Sold	Actual Bought
1	False	1	2	1000	1000	0	500
2	True	1	2	1000	1000	0	1000

Table 5.1: VerifyContract where the users have partially fulfilled the obligation and after fufillment

5.2 Performance

Due to the fact that the nodes are all running on the same machine, the runtimes mentioned below would represent the best case scenario whereas real life testing would include network delays. Since the test scenarios were performed on a private network, it was decided to utilize a spammer plugin. This adds an artificial network load to the IOTA tangle and can simulate a more real-world scenario. The specifications of the laptop and desktop are presented in Table 5.2.

	Desktop	Laptop
CPU	i9 12900k	i7 7500U
Cores/threads	16/16	2/4
Memory	16GB	24GB

Table 5.2: CPU and memory specifications of systems

The bar graph presented in Figure 5.9 compares the average runtime (in seconds) of the PlaceBids function when run on two different types of hardware: a laptop and a desktop computer. The testing was performed by creating a thread for each asset. The graph shows that the average runtime of the process increases as the number of assets being processed increases. When processing 1 asset, the average runtime on a laptop is 4.56 seconds and 4.83 seconds on a desktop, with the laptop being slightly faster. As the number of assets increases to 10, the average runtime on a laptop increases to 6.31 seconds and on a desktop it increases to 5 seconds, suggesting a better performance of the desktop. When processing 25 assets, the average runtime on a laptop is 9.27 seconds whereas on a desktop it is 6.92 seconds. The result shows a significant difference in performance between the two. Finally, when processing 50 assets, the average runtime on a laptop is 9.82 seconds and on a desktop is 9.31 seconds, with the desktop showing a slightly better performance. To sum up, the graph shows that the laptop performance is generally slower than the desktop but with a small difference when the number of assets is low. As the number of assets increases, the gap between the two performances broadens.

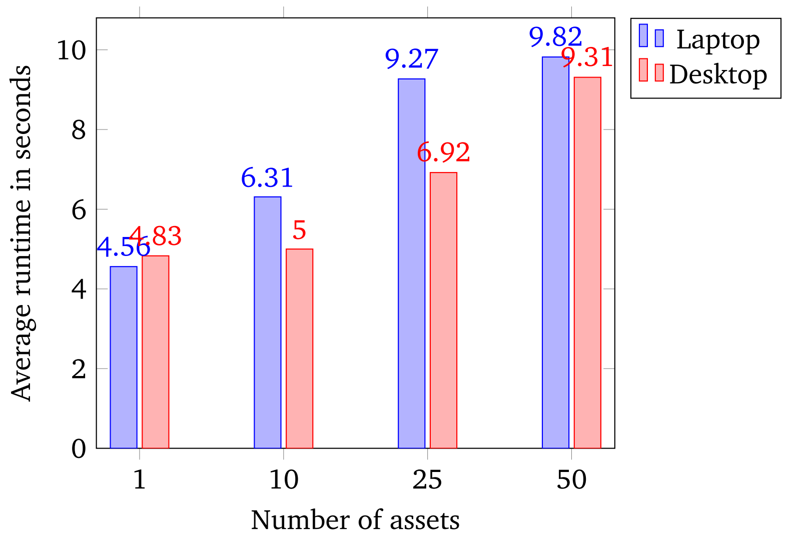


Figure 5.9: Runtime when running on different hardware

The bar graph presented in Figure 5.10 compares the average runtime (in seconds) for running a placebids function to the ledger with 1 and 4 nodes. The runtimes with 4 nodes utilized a quorum of 3, meaning that 3 out of the 4 nodes needed to agree to the proposed block. The graph shows that the average runtime of the process increases as the number of assets being processed increases, regardless of the number of nodes. When processing 1 asset, the average runtime on 1 node is 4.56 seconds and 12.08 seconds on 4 nodes, with the 1 node being significantly faster. As the number of assets increases to 10, the average runtime on 1 node increases to 6.31 seconds and on 4 nodes stays about the same, at 11.59 seconds, showing a better performance of 1 node. When processing 25 assets, the average runtime on 1 node is 9.27 seconds and on 4 nodes it is 13.97 seconds, showing a significant difference in performance between the two. Finally, when processing 50 assets, the average runtime on 1 node is 9.82 seconds and on 4 nodes is 15.10 seconds, with 1 node showing a slightly better performance. All in all, the graph shows that the 4-node performance is generally slower than 1-node performance, and the gap between the two performances widens as the number of assets increases.

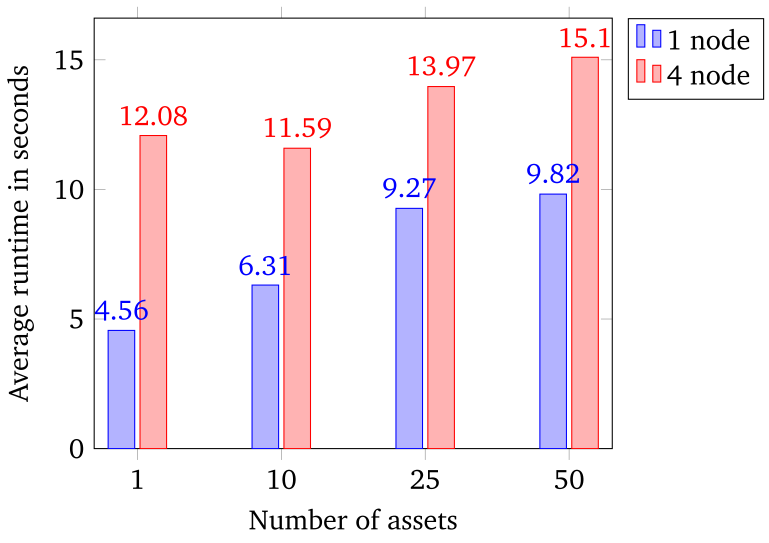


Figure 5.10: Runtime when different number of assets sends transactions

The relation between the amount of data present on the ledger and the runtime to call a view function is shown in Figure 5.11. The blue line represents the time for retrieving the last item in an array. It can be observed that as the number of elements increases, the time taken to process them also increases. When the number of elements is low, around 100-400, the time used to process them is relatively low, around 0.02-0.03 seconds. The time used to process the elements continues to increase as the number of elements increases, with time used reaching 0.27 seconds at 5000 elements. The red line represents the time used to retrieve information for a Mapping.

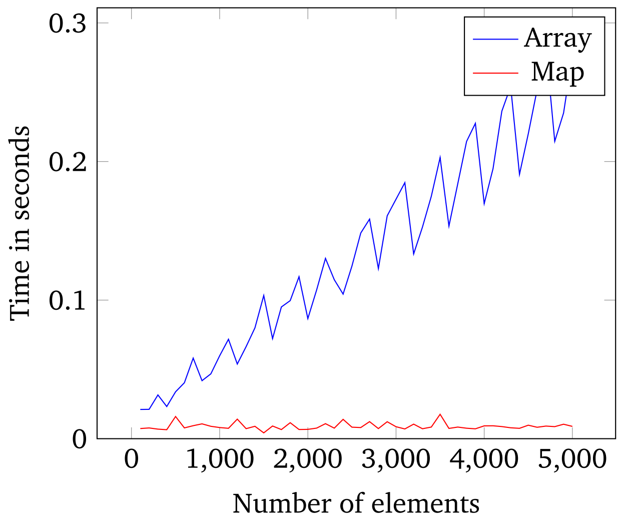


Figure 5.11: Time used as a function of the number of elements

To assess CPU and memory usage, the "psrecord" library was employed on a laptop equipped with four cores. During the test, the monitor was connected to the wasp chain process, and the test ran for around 70 seconds. The results, shown in Figure 5.12, reveal that the CPU usage readings could reach up to 400%. The test involved three blocks and three sets of transactions, with each set consisting of 50 assets and placing a bidding curve of 3 bids each.

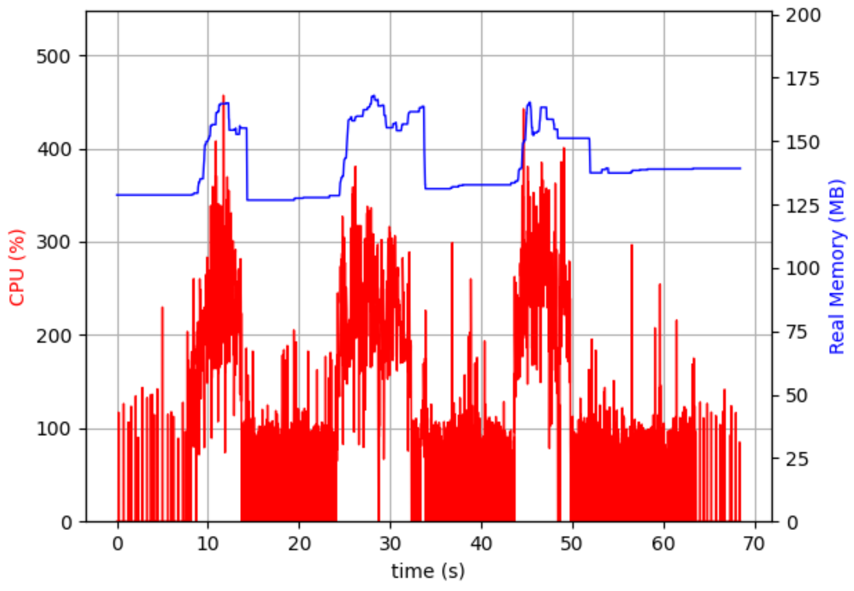


Figure 5.12: CPU and memory utilization of the wasp node on a laptop

The storage requirements were tested using 50 assets and executing 24 auctions rounds. This represented 24 hours with a time period of 1 hour. Each prosumer submitted a bidding curve consisting of 3 bids and the resulting contracts were also added. This resulted in an increase of the ledger size by **19 MB**. The Consumption log was also acquired using 50 prosumers and a 1 hour timeperiod, which resulted in required storage of **1.71 MB**.

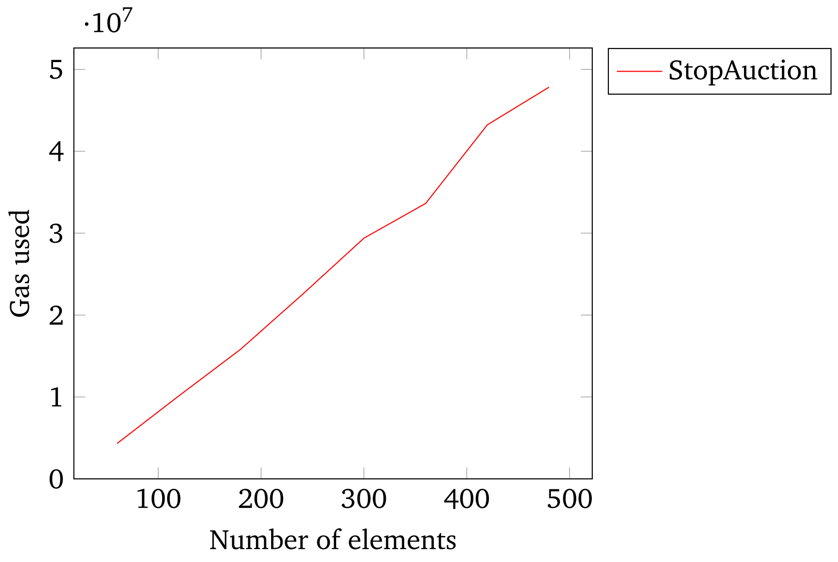


Figure 5.13: Gas used to perform the market matching based on number of bids placed

The StopAuction function performs the market matching of bids and offers. The number of bids and offers affect the amount of gas used as seen in Figure 5.13. The gas usage increased linearly, and 480 bids and offers resulted in a gas usage of 47826394; increasing the number of bids and offers to 540 resulted in an error, indicating that the requested operation exceeded the maximum gas limit allowed by the Virtual Machine:

```
{'code': -32000, 'message': 'request might require
more gas than it is allowed by the VM (50000000)'}
```

Chapter 6

Discussion

The purpose of the current study was to explore the possibility of utilizing the ISCP to provide a platform for energy markets. The results presented in Chapter 5 revealed interesting findings related to the performance of the wasp chain. This is one of the first studies that includes performance metrics in relation to energy market on ISCP. The discussion will be divided based on the research questions.

6.1 Is using IOTA smart contracts a feasible solution?

6.1.1 Research Question 1.1

Is logging data to the ledger possible within a suitable timeframe and storage possibility?

The runtime of the function to submit bids varied based on two factors: the number of assets sending transactions and the number of nodes in the network. Additionally, the runtimes on both desktop and laptop increased with the number of assets sending transactions. The data indicates that there was no limit to the number of transactions the chain could handle within the test parameters presented in this study. These findings align with previous research on different DLTs, such as the 12-second transaction time achieved in [24] with 28 nodes and 64 transactions per second, and the 15-second transaction time achieved in [37] with 5 photovoltaic agents in their microgrids. However, the scalability of DLT-based solutions like the tested wasp chain implementation may be lower than those that use a database as a second layer solution, as seen in studies such as [22],[23], [30]. Consequently, future large-scale implementations using the wasp chain may encounter similar limitations as other DLTs, highlighting the importance of limiting the number of assets per wasp chain to eliminate the need for implementing database layers.

The transaction time increased with a higher number of assets submitting bids to the platform. Nordpool offers trading in 15-minute, 30-minute and hourly blocks to meet the needs of different market areas. The average execution time for a transaction was between 4.65 and 12.08 seconds for a single address placing bids. If the worst case transaction time is 12.08 seconds and the auction round is 15 minutes, each asset can provide around 75 bids. Hence, this seems sufficient when assets submit bids and offers for their individual consumption and production. Nevertheless, this can lead to complications in the setting of a centralized provider for bids for the user. If the centralized bidder is required to add bids for each individual user, this may cause difficulties in delivering all bids within the required timeframe. A central entity submitting bids and offers for 100 users would require between 7.6 minutes and 20.1 minutes. This can be satisfactory for auction rounds in the 1-hour mark, but can cause difficulties with shorter auction rounds or more users. This can be mitigated by committing bids for multiple users simultaneously, but it clearly shows that the specific implementations in smart contracts need to be taken into account.

In addition, the runtimes seem to correlate with an increase in the number of nodes in the committee. This observation suggests that there is a trade-off between the number of nodes and the runtime. One possible explanation for this trend is that as the number of nodes increases, the system requires more time to reach a consensus on the transactions. This study has some limitations as it only evaluated the performance of 1 and 4 nodes in the network. However, the results obtained are still meaningful and can provide insights into the performance of the system. Furthermore, the results obtained are comparable to the findings in [38], which evaluated 5 and 10 nodes in the network. In the case where the market operator is responsible for all the nodes, the primary advantage of a larger number of nodes is improved availability in the event of node disconnection. One potential direction for future research is investigating the possibility of having each asset host its own node, which would enable a significant increase in the number of nodes.

It should be noted that the test setup used to execute the experiments relating to the laptop runtime presented in Chapter 5 might have some limitations. As shown in Figure 5.12, the laptop, equipped with four cores, was nearly fully utilized during the tests. Moreover, all the wasp chain nodes and the hornet nodes that make up the IOTA tangle were run on the same computer. The number of cores is related to the number of nodes in the committee, with each node assigned at least one core. However, this factor may have influenced all results and should be taken into consideration when analyzing the outcomes of the results.

Furthermore, it is important to note that the runtimes mentioned earlier were conducted on a single computer and did not include network delays. To improve this, the possibility of multiple wasp chains anchoring on the tangle makes it possible to distribute assets into different wasp chains based on their physical location, whether it is a neighbourhood or a city district. This approach could potentially benefit the wasp chain by utilizing the proximity of nodes to potentially reduce network delays. As demonstrated in [26], sharding using the ChainSpace Hyperledger showed improvements in throughput, indicating that further examination of this strategy for the wasp chain may be worthwhile. Additionally, this is relevant to energy markets, as balancing energy based on location is crucial. Therefore, the wasp chain can potentially improve its performance by utilizing distributed computing and sharding to optimize its network and improve its efficiency.

The microgrids can be run in parallel since a group of independent wasp nodes can control each microgrid, as shown in Figure 6.1. This allows for standalone implementations that can be written independently of the entire network. Each microgrid has its own instance of the wasp chain that can be called without the knowledge of the other chains.

Another advantage of using the tangle is the independence of anchoring to the network, unlike Layer 1 Ethereum solutions, where transactions are dependent on network traffic. Therefore, the use of the public tangle is unlikely to impact the transaction confirmation time. In essence, it should be noted that each microgrid has its own performance limits, but the advantage of running them in parallel is that the global ledger is not significantly influenced by the increase in the number of microgrids. This means that the scalability of the system can be maintained even as the number of microgrids grows, provided that the performance limits of each microgrid are taken into consideration.

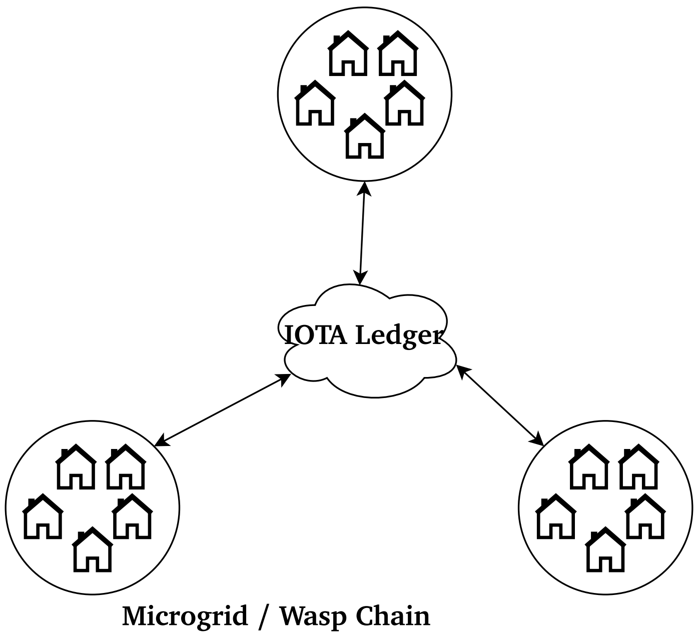


Figure 6.1: Overview of microgrids

The storage requirements were tested using 50 prosumers and an 1-hour time period for 24 hours. Each prosumer submitted a bidding curve consisting of 3 bids, and the resulting contracts were also added. This resulted in an increase of the ledger size by **19 MB**. The Consumption log was also acquired using 50 prosumers and a 1-hour time period, and this resulted in the required storage of **1.71 MB**. Extrapolating this information to a one-year period yields a storage requirement of approximately 6.9 GB for bids and 625 MB for the consumption log. An increase in the number of nodes from the perspective of the wasp chain would result in the duplication of data, whereas from the perspective of the tangle, the data is stored off-ledger and does not significantly impact the network. It is worth mentioning that as the system grows, additional testing and optimization may be required to ensure efficient and effective data management.

In this project a role-based access control was implemented to control access to adding information to the ledger. However, it should be noted that this does not prevent access to the information on the ledger. The choice of the type of distributed ledger, whether a public or private ledger, is a tradeoff between privacy and openness. The large amount of data generated by the microgrids can be a privacy concern, as highlighted in the work of [23], where a malicious agent can establish a relationship between consumer and producer data and their identity. Due to the small size of microgrids, the possibility of establishing such relationships may be increased. This can be mitigated by preventing the storage of private information, as proposed in the work of [23] or [34]. However, using smart contracts to execute market matching and contract verification can pose challenges to encrypting or obfuscating data, as the information needs to be available to all nodes but this could hinder the application of encryption techniques. Many solutions address this issue by utilizing private networks, thus limiting the openness of the information. The wasp chain which utilizes a private network anchoring to the public network limits the verifiability for end-users as they only have access to the anchor hash in the public tangle. As the wasp chain consists of key/value pairs and does not use a blockchain, the information can be pruned, allowing for the removal of data about customers who wish to be forgotten.

6.1.2 Research Question 1.2

Can auction round be performed on the ledger?

To ensure the integrity of the Layer 2 implementation and prevent malicious actors, it is beneficial to include a cost associated with sending transactions, even though it is not strictly necessary. Additionally, because the computations are distributed, optimizing code to reduce gas usage can ultimately lead to less computational power required in the network. The default gas limit per block set is 15M. Based on the analysis of the committing of bids, it was determined that each bid required approximately 100 000 gas. With a block time of 5 seconds, this corresponds to a theoretical throughput of 30 TPS and 150 bids per block. During testing, the achieved TPS was found to be between 9.9 and 32.9, which is within the expected range of the theoretical values. When comparing to the private network utilized in [22], they had set a block time of 15 seconds and each logging consumed 137,131 gas which resulted in 4 transactions per minute.

A simple uniform double market solution was used to match the bids and offers. However, the bidding algorithm and the trading algorithm could be more advanced to find a better solution, such as [37], which looks at the users and system benefits. This was, however, not in the scope of this study. The current solution utilized the last traded value as the price to settle all trades, but this may not be sufficient as the result may not be representative of the correct clearing price.

The market matching algorithm was limited to a maximum between 480-540 bids per auction round, due to the increasing amount of computation, although this is a limit which can be modified. Nevertheless, a transaction may use bandwidth, storage and computation in arbitrary quantities. Not capping the computation may cause halting problems and result in denial-of-service attacks. Since the definition of flexibility is that there are multiple possibilities to reduce or increase consumption based on multiple scenarios, it may be assumed that the number of bids per asset is more than 1. With the current setup of the wasp chain, depending on the number of bids per asset, the number of possible assets to maintain in each wasp chain is shown in Figure 6.1.

# of bids per asset	maximum # of assets
1	480
2	240
3	160
4	120
5	96
10	48
20	25

Table 6.1: Number of assets depending on number of bids.

This adds to the issue that smart contracts are not particularly ”smart”. This is due to the computational cost of executing complex code on a DL and because programming languages are often restricted due to computational and security reasons. However, it is important to be aware that complex computation on the EVM does decrease the amount of data possible to add [41]. Thus the appropriate size for the wasp network is related to the computations which are performed on the network. The effects of increasing the gas limit per block need to be explored. However, decreasing the number of assets per wasp chain and increasing the number of wasp chains increases the maintenance of the networks. The limited number of bids which can be processed within a smart contract emphasises the possibility to utilize the DL as a data storage solution to verify data instead of executing computations.

The IOTA tangle offers a significant benefit over other blockchain platforms by being fee-less. As highlighted by [25], using the public Ethereum network can be costly, while the IOTA network allows for anchoring the state with little to no cost. This is a benefit since even second-layer solutions need to anchor the state on the layer 1 ledger. Ethereum has since switched to a PoS consensus algorithm, which reduces the computational power needed to run the network, but this does not necessarily translate to improved scalability or reduced costs. However, it should be noted that the IOTA tangle may not be affected by the consensus algorithm in the same way as Ethereum and Bitcoin, as the mechanisms for adding transactions to the ”Tangle” are different. Although the computational requirements are lower, the sequential nature of the blockchain layered on top of the tangle may impact the network’s throughput. Most of the solutions found in the SLR deployed a private ledger, but the wasp chain stands out by enabling a combination of public and private ledgers. Keeping information private while still anchoring the state on the public ledger makes it possible to increase trust in the network. Additionally, the wasp chain can be transformed into a consortium network if desired. However, this would make all information public and only restrict the ability to create and add transactions to a selected group of participants. Such a network configuration may be appropriate for certain use cases, but it is important to carefully consider the trade-offs between privacy and accessibility when making such a decision.

6.1.3 Research Question 1.3

Can verification of contracts be performed on the ledger?

When working with energy markets, verification of contracts on the ledger is an important consideration. This process involves the utilization of large amounts of data. As a result, it is essential to consider the efficiency of information retrieval when implementing data structures to ensure timely and accurate contract verification. The efficiency of retrieving information from the wasp chain depends largely on the data structure of the stored information.

A mapping has a lookup time of O(1), which means that the retrieval time is constant and independent of the number of elements stored in the data structure. On the other hand, the lookup time for an array can be O(log n) or O(n), depending on whether it is sorted or unsorted. The reason for this is that a sorted array can perform a binary search, which is more efficient than a linear search performed in an unsorted array. If the data stored in the array is a time series, it can be assumed to be almost sorted. In such cases, using a sorting algorithm, which has a runtime of O(nlog n) [41], may not be necessary. Instead, a linear search through the array may be sufficient to find the desired element. Therefore, it is crucial to evaluate each data structure based on how the data is expected to be used. As the data in the wasp chain is stored in a key/value pair, there is no inherent data structure for performing searches, and it must be implemented separately. The two types of data structure seem to have certain advantages based on the design and how it is decided to log information. Double mapping is particularly useful for storing structured information that corresponds to a specific auction round. This mapping type allows for efficient retrieval of information based on the auction round and the user ID. On the other hand, an array data structure is more appropriate for logging sporadic events that may not match the auction rounds. This type of data structure is useful when information is logged at irregular intervals.

As demonstrated in this study, when logging the consumption data of a single asset every hour for a year results in 8,760 elements. Extrapolating this data to a 15-minute time block results in 35,040 elements and an estimated worst-case retrieval time of 1.7 seconds. Therefore, it is advisable to keep the logging of consumption and production to the same time frame as the auction period to reduce the lookup time. Comparing a single log to the generated contracts is much simpler, which greatly reduces the lookup time.

The mentioned number of elements is per asset, which raises the subject of how the assets are divided. Dividing the components into smaller assets can improve traceability and provide a more detailed view of consumption and production. The study referenced in [22] was able to achieve 5 transactions per second per user for up to 2000 users, which may not be feasible with the ISCP implementation presented in this study. Therefore, the use of aggregation may be required to be possible within the time frame, but this presents the problem of storing the information outside of the blockchain. The solution may need to implement an aggregator that combines the information to ensure that a single asset does not exceed the maximum number of transactions per second. For example, if a building has a photovoltaic energy source, battery storage, and multiple flexible energy consumers, they can be treated as one asset with a combined log of consumption or production. However, since the contracts are generated per bid in the bidding curve, keeping track of which bidding in the curve that each asset has posted may be beneficial.

It is also important to mention that while anchoring the state on the tangle provides immutability to the data, it does not automatically verify its accuracy or authenticity. Therefore, an additional layer of authentication is required to ensure that the data added to the blockchain is trustworthy [22, 30]. It is important to understand that the immutability of data on the ledger does not guarantee its trustworthiness and that incorrect or false data can still be added to the blockchain. Hence, proper measures must be taken to authenticate the data before it is added to the ledger.

6.2 Are there other road blocks for the implementation

Initially, it was planned to utilize the native Rust and Go implementation to create smart contracts but due to some early struggle, it was decided to coose a Solidity implementation. The ISCP is relatively new and due to this, there is limited knowledge available on published code. Most of the information surrounding ISCP was quite basic and did not provide in-depth knowledge of debugging and writing code. However, knowledge will undoubtedly improve as the technology matures. However, the implementation currently cannot support native functions that IOTA may offer. Furthermore, operating the wasp chain in its newly released state can be a challenging process, which may lead to more errors.

The Solidity programming language and EVM also have their own set of challenges. Debugging a smart contract using Remix IDE is somewhat restricted, and identifying where the transaction failed can be a difficult process. In addition, smart contracts are immutable, meaning that any protocol for changing the contract must be established beforehand. Any errors within the smart contract could have significant effects on the system and would be difficult to repair due to the irreversible nature of the contract. However, by using a blockchain-based solution, it is possible to save the state of the data before errors occur, allowing for the system to be restored to a previous state if needed.

In the current regulatory framework, the DSO controls the electricity market as they are responsible for the grid. The current goal of the system operator is to keep the grid operational and provide benefits for all users on the network. However, regulatory changes must occur to enable users to play a more active role in trading by utilizing DLTs. Current regulations do not favour individual independence [42]. The core of DLT lies in establishing mutual trust in decentralized and autonomous systems. However, the number of nodes and the number of responsible entities dictates how decentralized the solution is, and this has some caveats in an energy market where the DSO is solely responsible for all the nodes. In addition, the proposed system in its current state has a centralized estimation algorithm, which may provide a single point of failure and can be particularly concerning. However, even with centralized estimation, multiple nodes may aid the system if one or a few nodes lose connection.

Chapter 7

Conclusion and future work

This study has presented the required knowledge and current solutions to ensure scalability for using DLT in local flexibility markets. Additionally, the technical feasibility of an ISCP solution has been explored and compared to current state-of-the-art solutions and the ISCP scalability potential has been illuminated.

This study is a significant contribution to the field of decentralised energy markets as it is one of the first to examine the potential of ISCP. The findings indicate that additional research is required to thoroughly investigate the potential of this field and to develop new technologies and applications based on the preliminary investigations.

To facilitate an energy market on DLTs, consumption and production data must be stored, bids and offers must be placed, contracts must be generated between buyers and sellers, and contracts must be compared to actual consumption and production. However, a decentralised energy market does not inherently require all three aspects to operate. Throughout the three iterations, the system met the defined requirements; however, the evaluation of the prototype revealed some limitations.

The design science approach was used to develop and evaluate the solution. The methodology involved the creation of an artifact which was rigorously evaluated through various methods. Through this approach, the thesis provides a knowledge contribution in the IOTA/energy markets field. In order to create the energy market, a private IOTA network was set up and a WASP chain was launched, followed by the development of a smart contract.

The solution has satisfied the main requirements of the project and proof of concept has been developed. However, the system has not been tested by other developers or in a real-life test, so it is difficult to conclude that the solution will work in real life. Despite the limitations, findings from the present study can help understand practices and develop approaches to the new ISCP. The results of this study may be useful in the scope of solutions and help DSO become more aware of possibilities to include the users to take a larger role in the energy market.

The ability to divide the network into multiple smaller chains appears advantageous and can assist in the scalability of the solution, thus further reducing its performance requirements. The performance of the proposed solution can be divided into two categories: limitations imposed by the design of the ledger technologies and limitations imposed by the optimization of the smart contract.

Is storing data in the ledger possible within a suitable time frame and storage possibility?

RQ 1.1 is mostly influenced by the performance limitations of the design of the DLT. Depending on variables such as the number of nodes in the network and the hardware the nodes were operating on, the total execution time to confirm a transaction ranged from 4.56 to 15.1 seconds. This is consistent with other studies regarding other single-layer solutions and appears to meet the requirements of energy markets, but it does not match the performance of other second-layer solutions. The theoretical 30 TPS for bids and the measured 32.9 to 9.9 is based on multiple variables and it is difficult to specify the time complexity of the solution based on the number of assets and nodes because the scope of the study was rather small. However, it is worth noting that this performance applies to each Local Flexibility market. This significantly increases the scalability by allowing each Local flexibility market to process close to the amount that the global public Ethereum network can handle.

However, one could argue that distributed ledger technologies are essentially storage solutions that have no bearing on the authenticity of the represented data. The DLT guarantees the verifiability of all data sent to it. However, inaccurate data remains inaccurate on the DLT. A system deploying DLTs would require a method for authenticating bids, offers, and, most importantly, consumption and production data.

Can auctions be performed on the ledger?

RQ1.2 was mostly influenced by the performance limitations of the smart contract. With the implemented matching algorithm, the maximum number of bids per auction round was between 480 and 540. This is a limit that can be altered by modifying the configuration of the wasp chain, and various matching algorithms may use varying amounts of gas and permit fewer or more bids per auction round, but this must be tested further. A simplistic uniform double auction was used in the setup. However, due to time constraints, the platform implementation took priority. As the platform has now been verified, further development can be conducted to examine alternative market clearing algorithms.

Can verification of contracts be performed on the ledger?

RQ1.3 was affected by both the performance limitations of the smart contract and by the performance limitations of the design of the DLT. The structures generated by the smart contract have a significant impact on performance. This was demonstrated in this thesis through a relatively straightforward comparison between an

Array and a Mapping (Hash Map), but the comparison shows that a database solution can have a significantly greater number of functions than smart contracts. In addition, the proposed solution is limited in comparison to database layer solutions based on the number of assets and the frequency with which information is stored.

Are there other roadblocks to the implementation of DLT?

Although the implementation of native functions offered by IOTA has been challenging due to the limited knowledge about coding for the relatively new ISCP, continuous improvement efforts are expected to overcome these difficulties as knowledge about ISCP expands. Solidity and the EVM implementation on the wasp chain were utilized early on to address the challenge of programming in Rust/Go.

In addition, the necessity of implementing DLT in a centralized energy market, where the DSO is responsible for all nodes, raises questions about establishing mutual trust in decentralized systems. Furthermore, the current centralized system may be vulnerable to a single point of failure, especially if energy consumption estimation remains centralized. To address this issue, a decentralized estimation system with multiple nodes could potentially provide greater resilience to the system in the event of a node failure. As DLT continues to evolve, it offers promising solutions for creating more resilient and trustworthy decentralized energy systems.

7.1 Future work

This study provides useful insights into the potential use of the ISCP framework in energy markets, and there are still many areas for future work. Nevertheless, this study serves as an excellent foundation for the future development of energy markets on ISCP. In particular, it is crucial to investigate the performance of the wasp chain, which serves as the foundation of the energy market platform. While the experiments conducted in this study were informative, further comprehensive testing is necessary to gain a more in-depth understanding of the capabilities and limitations of the wasp chain. Further investigation of the wasp chain performance and its impact on energy markets is necessary to identify areas for optimization and improvement. In particular, it will be important to evaluate the performance of the wasp chain under different conditions, such as high transaction volumes or higher number of nodes. In addition, there needs to be further investigation of the potential benefits of implementing decentralized bidding algorithms. This will help to identify potential bottlenecks or areas for optimization, which can in turn improve the efficiency and reliability of the energy market platform.

Bibliography

[1] G. Strbac, 'Demand side management: Benefits and challenges,' *Energy Policy*, vol. 36, no. 12, pp. 4419–4426, 2008, Foresight Sustainable Energy Management and the Built Environment Project, ISSN: 0301-4215.

[2] Y. Xinyi, Z. Yi and Y. He, 'Technical characteristics and model of blockchain,' in *2018 10th International Conference on Communication Software and Networks (ICCSN)*, 2018, pp. 562–566.

[3] W. Li and M. He, 'Comparative analysis of bitcoin, ethereum, and libra,' in *2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS)*, 2020, pp. 545–550.

[4] S. Paavolainen and C. Carr, 'Security properties of light clients on the ethereum blockchain,' *IEEE Access*, vol. 8, pp. 124 339–124 358, 2020.

[5] E. Drąsutis, 'Iota smart contracts,' 2021.

[6] J. O. Moltu, 'Implementations of distributed ledger in demand response applications,' 2022.

[7] N. Renugadevi, S. Saravanan and C. Naga Sudha, 'Iot based smart energy grid for sustainable cites,' *Materials Today: Proceedings*, 2021, ISSN: 2214-7853.

[8] C. W. Gellings and J. H. Chamberlin, 'Demand-side management: Concepts and methods,' Jan. 1987.

[9] S. S. Torbaghan, N. Blaauwbroek, P. Nguyen and M. Gibescu, 'Local market framework for exploiting flexibility from the end users,' in *2016 13th International Conference on the European Energy Market (EEM)*, 2016, pp. 1–6.

[10] T. Capper, A. Gorbatcheva, M. A. Mustafa, M. Bahloul, J. M. Schwidtal, R. Chitchyan, M. Andoni, V. Robu, M. Montakhabi, I. J. Scott, C. Francis, T. Mbavarira, J. M. Espana and L. Kiesling, 'Peer-to-peer, community self-consumption, and transactive energy: A systematic literature review of local energy market models,' *Renewable and Sustainable Energy Reviews*, vol. 162, p. 112 403, 2022, ISSN: 1364-0321.

[11] S. Nakamoto, 'Bitcoin: A peer-to-peer electronic cash system,' May 2009. [Online]. Available: `http://www.bitcoin.org/bitcoin.pdf`.

[12] V. Buterin, 'Ethereum white paper: A next generation smart contract & decentralized application platform,' 2013.

[13] S. Popov, 'The tangle,' *White paper*, vol. 1, no. 3, p. 30, 2018.

[14] 'The tangle.' (2022), [Online]. Available: `https://wiki.iota.org/learn/about-iota/tangle`.

[15] 'Dag the dlt! directed acyclic graph for enterprise blockchain!' (2019), [Online]. Available: `https://www.cbcamerica.org/blockchain-insights/dag-the-dlt-directed-acyclic-graph-for-enterprise-blockchain`.

[16] L. Lamport, R. Shostak and M. Pease, 'The byzantine generals problem,' in *Concurrency: The Works of Leslie Lamport*. New York, NY, USA: Association for Computing Machinery, 2019, pp. 203–226, ISBN: 9781450372701.

[17] C. Stoll, L. Klaaßen and U. Gallersdörfer, 'The carbon footprint of bitcoin,' *Joule*, vol. 3, no. 7, pp. 1647–1661, 2019, ISSN: 2542-4351.

[18] N. Sealey, A. Aijaz and B. Holden, *Iota tangle 2.0: Toward a scalable, decentralized, smart, and autonomous iot ecosystem*, 2022. DOI: `10.48550/ARXIV.2209.04959`.

[19] 'The merge: Implications on the electricity consumption and carbon footprint of the ethereum network.' (2022), [Online]. Available: `https://carbon-ratings.com/eth-report-2022`.

[20] V. Buterin *et al.*, 'A next-generation smart contract and decentralized application platform,' *white paper*, vol. 3, no. 37, pp. 2–1, 2014.

[21] T. Chen, X. Li, Y. Wang, J. Chen, Z. Li, X. Luo, M. H. Au and X. Zhang, 'An adaptive gas cost mechanism for ethereum to defend against under-priced dos attacks,' Dec. 2017, ISBN: 978-3-319-72358-7.

[22] C. Pop, M. Antal, T. Cioara, I. Anghel, D. Sera, I. Salomie, G. Raveduto, D. Ziu, V. Croce and M. Bertoncini, 'Blockchain-based scalable and tamper-evident solution for registering energy data,' *Sensors*, vol. 19, no. 14, 2019, ISSN: 1424-8220.

[23] C. D. Pop, M. Antal, T. Cioara, I. Anghel and I. Salomie, 'Blockchain and demand response: Zero-knowledge proofs for energy transactions privacy,' *Sensors*, vol. 20, no. 19, 2020, ISSN: 1424-8220.

[24] A. Lucas, D. Geneiatakis, Y. Soupionis, I. Nai-Fovino and E. Kotsakis, 'Blockchain technology applied to energy demand response service tracking and data sharing,' *Energies*, vol. 14, no. 7, 2021, ISSN: 1996-1073.

[25] M. Foti and M. Vavalis, 'Blockchain based uniform price double auctions for energy markets,' *Applied Energy*, vol. 254, p. 113 604, 2019, ISSN: 0306-2619.

[26] S. Mitra, K. Joshi and K. Ramamritham, 'Centra: City-scale energy transaction application for the smart grid,' in *2020 IEEE International Conference on Power Systems Technology (POWERCON)*, 2020, pp. 1–6.

[27] S. M. H. Bamakan, A. Motavali and A. Babaei Bondarti, 'A survey of blockchain consensus algorithms performance evaluation criteria,' *Expert Systems with Applications*, vol. 154, p. 113 385, 2020, ISSN: 0957-4174.

[28] C. Pop, T. Cioara, M. Antal, I. Anghel, I. Salomie and M. Bertoncini, 'Blockchain based decentralized management of demand response programs in smart energy grids,' *Sensors*, vol. 18, no. 1, 2018, ISSN: 1424-8220.

[29] R. Rahmani and Y. Li, 'A scalable digital infrastructure for sustainable energy grid enabled by distributed ledger technology,' *Journal of Ubiquitous Systems Pervasive Networks*, vol. 12, pp. 17–24, Mar. 2020.

[30] S. Saha, N. Ravi, K. Hreinsson, J. Baek, A. Scaglione and N. G. Johnson, 'A secure distributed ledger for transactive energy: The electron volt exchange (eve) blockchain,' *Applied Energy*, vol. 282, p. 116 208, 2021, ISSN: 0306-2619.

[31] A. Tomar and S. Tripathi, 'Blockchain-assisted authentication and key agreement scheme for fog-based smart grid,' *Cluster Computing*, Sep. 2021, ISSN: 1573-7543.

[32] Q. Yang, H. Wang, T. Wang, S. Zhang, X. Wu and H. Wang, 'Blockchain-based decentralized energy management platform for residential distributed energy resources in a virtual power plant,' *Applied Energy*, vol. 294, p. 117 026, 2021.

[33] C. Patsonakis, S. Terzi, I. Moschos, D. Ioannidis, K. Votis and D. Tzovaras, 'Permissioned blockchains and virtual nodes for reinforcing trust between aggregators and prosumers in energy demand response scenarios,' in *2019 IEEE International Conference on Environment and Electrical Engineering and 2019 IEEE Industrial and Commercial Power Systems Europe (EEEIC / I CPS Europe)*, 2019, pp. 1–6.

[34] A. Mandal, 'On security and privacy of consensus-based protocols in blockchain and smart grid,' English, Ph.D. dissertation, Mannheim, 2020.

[35] K. Peffers, T. Tuunanen, M. A. Rothenberger and S. Chatterjee, 'A design science research methodology for information systems research,' *Journal of management information systems*, vol. 24, no. 3, pp. 45–77, 2007.

[36] A. R. Hevner, S. T. March, J. Park and S. Ram, 'Design science in information systems research,' *MIS quarterly*, pp. 75–105, 2004.

[37] C. Mullaney, A. Aijaz, N. Sealey and B. Holden, *Peer-to-peer energy trading meets iota: Toward a scalable, low-cost, and efficient trading system*, 2022.

[38] J. Rosenberger, F. Rauterberg and D. Schramm, 'Performance study on iota chrysalis and coordicide in the industrial internet of things,' in *2021 IEEE Global Conference on Artificial Intelligence and Internet of Things (GCAIoT)*, 2021, pp. 88–93.

[39] S. Hackett, B. Kvaal and M. Fure, *D2. 3: Report on the flexibility market*, 2019.

[40] S. Danielsen, K. Livik, B. O. Berthelsen and G. Sizov, *D5.5: Energy trading market demonstration*, 2019.

[41] I. S. Ochôa, R. A. Piemontez, L. A. Martins, V. R. Q. Leithardt and C. A. Zeferino, 'Experimental analysis of the processing cost of ethereum blockchain in a private network,' 2019.

[42] M. Maldet, F. H. Revheim, D. Schwabeneder, G. Lettner, P. C. del Granado, A. Saif, M. Löschenbrand and S. Khadem, 'Trends in local electricity market design: Regulatory barriers and the role of grid tariffs,' *Journal of Cleaner Production*, vol. 358, p. 131 805, 2022, ISSN: 0959-6526.

Appendix A

Additional Material

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.13;
contract Mapping {
    mapping(uint => Auction) public auctions;
    uint[] auctionIds;
    mapping(address => bool) whitelistedAddresses;
    uint public currentAuction = 0 ;

    address public owner;
    struct Bid{
        uint user;
        int quantity;
        int price;
        bool isSelling;
        int bidNr;
    }
    struct Auction{
        uint auctionId;
        Bid[] bids;
        Bid[] offers;
        Contract[] contracts;
        bool finished;
        uint startTime;
        uint endTime;
        Clearing clearing;
    }
    struct Contract{
        uint buyId;
        uint sellId;
        int quantity;
        int price;
        int actualSold;
        int actualBought;
    }
    struct Clearing {
        int clearingQuantity;
        int clearingPrice;
    }

    mapping(uint => Log[]) public logging;

    struct Log {
        uint userId;
        int usage;
        bool isSelling;
        uint timeStamp;
    }


    mapping(uint => mapping(uint => Log)) public logMapping;
    uint[] timeStamps;

    function logUsageMapping(uint userid, int usage, bool isselling, uint timestamp) public {
        logMapping[userid][timestamp]=(Log(userid, usage,isselling,timestamp));
    }

    function getUsageMap( uint userId, uint timeframe) public view returns (Log memory log){
        return logMapping[userId][timeframe];

    }
    function getUsage( uint userId, uint timeframe) public view returns (Log memory log){
        for (uint i=0;i<logging[userId].length;i++){
                if (logging[userId][i].timeStamp == timeframe){
                    return logging[userId][i];
                }

        }
        return Log(0, 0,false,0);
    }
    function getUsageInv( uint userId, uint timeframe) public view returns (Log memory log){
        for (uint i=logging[userId].length;i>=0;i--){
                if (logging[userId][i].timeStamp == timeframe){
                    return logging[userId][i];
                }

        }
        return Log(0, 0,false,0);
    }

    function logUsage(uint userid, int usage, bool isselling, uint timestamp) public {
        logging[userid].push(Log(userid, usage,isselling,timestamp));
        //check if
    }
    function logUsages(Log[] memory _logs) public {
        for (uint i=0;i<_logs.length;i++){
            logging[_logs[i].userId].push(Log(_logs[i].userId, _logs[i].usage,_logs[i].isSelling,_logs[i].timeStamp));
        }
        //check if
    }
    function logUsagesMapping(Log[] memory _logs) public {
        for (uint i=0;i<_logs.length;i++){
            logMapping[_logs[i].userId][_logs[i].timeStamp]=(Log(_logs[i].userId,_logs[i].usage,_logs[i].isSelling,_logs[i].timeStamp));
        }
    }
    constructor(){
        owner = msg.sender;
        whitelistedAddresses[msg.sender] = true ;
    }
    function changeOwner(address newOwner) public onlyOwner {
        owner = newOwner;
        whitelistedAddresses[newOwner] = true;

    }

    function getCurrentAuction() public view returns (uint current){
        return currentAuction;
    }
    function auctionFinished() public view returns (bool finished){
        return auctions[currentAuction].finished;
    }
    modifier onlyOwner() {
        require(msg.sender == owner, "Ownable: caller is not the owner");
        _;
    }
    modifier isWhitelisted(address _address) {
        require(whitelistedAddresses[_address], "You need to be whitelisted");
        _;
    }

    function verifyUser(address _whitelistedAddress) public view returns(bool) {
        bool userIsWhitelisted = whitelistedAddresses[_whitelistedAddress];
        return userIsWhitelisted;
    }

    function addUser(address _addressToWhitelist) public onlyOwner {
        whitelistedAddresses[_addressToWhitelist] = true;
    }
    event auctionStarted(uint _auctionId);
    event auctionStopped(uint _auctionId);

    function newAuction(uint _auctionId, uint startTime, uint endTime) public onlyOwner {
        if (auctions[currentAuction].finished || currentAuction==0){
            if( currentAuction==0 || auctions[_auctionId].auctionId == 0){
                Auction storage a = auctions[_auctionId];
                a.auctionId = _auctionId;
                a.finished =false;
                a.startTime = startTime;
                a.endTime = endTime;
                currentAuction = _auctionId;
                emit auctionStarted(currentAuction);
            }

        }
    }

    function stopAuction() public onlyOwner {
        if(!auctions[currentAuction].finished){
            auctions[currentAuction].finished = true;
            computeClearing();
            emit auctionStopped(currentAuction);
        }

    }

    function getAuction(uint _auctionId) public view returns (uint auctionId/*,
        uint quantity,
        uint price,
        bool isSelling,
        uint bidNr*/) {
        // Mapping always returns a value.
        // If the value was never set, it will return the default value.
        return auctions[_auctionId].auctionId;
    }
    function placeMultBids(Bid[] memory _bids) public isWhitelisted(msg.sender){
        if (!auctions[currentAuction].finished){
            for (uint i=0;i<_bids.length;i++){
                if(_bids[i].isSelling){
                    auctions[currentAuction].offers.push(_bids[i]);
                }else{
                    auctions[currentAuction].bids.push(_bids[i]);
                }
            }
        }
    }
    function placeBid(uint _user, int _quantity, int _price, bool _isSelling, int _bidNr) public isWhitelisted(msg.sender){
        if (!auctions[currentAuction].finished){
            if(_isSelling){
                auctions[currentAuction].offers.push(Bid(_user,_quantity, _price, _isSelling,_bidNr));
            }else{
                auctions[currentAuction].bids.push(Bid(_user,_quantity, _price, _isSelling,_bidNr));
            }
        }
    }

    function getBids(uint _auctionId) public view returns (Bid[] memory){
        return auctions[_auctionId].bids;
    }

    function getOffers(uint _auctionId) public view returns (Bid[] memory){
        return auctions[_auctionId].offers;
    }

    function verifyContract(uint _auctionId, uint _userId) public view returns (
         bool, Contract[] memory){
        int amountBought;
        int amountSold;
        Log[] memory logForUser;
        logForUser = logging[_userId];
        Contract[] memory contractsforUser = auctions[_auctionId].contracts;
        bool fulfilled = false;
        for (uint i=0;i<logging[_userId].length;i++){
            if (logging[_userId][i].timeStamp > auctions[_auctionId].startTime &&
                 logging[_userId][i].timeStamp < auctions[_auctionId].endTime){
                //if buyid or sellid == userid -> save
                //logForUser.push(logging[_userId][i])
                if (logging[_userId][i].isSelling){
                    amountSold += logging[_userId][i].usage;
                }
                else {
                    amountBought += logging[_userId][i].usage;
                }

            }
        }
        for (uint i = 0; i < contractsforUser.length; i++) {
                    if (contractsforUser[i].buyId == _userId) {
                        if (amountBought >= contractsforUser[i].quantity) {
                            contractsforUser[i].actualBought = contractsforUser[i].
                                 quantity;
                            amountBought -= contractsforUser[i].quantity;
                            fulfilled = true;
                        }else{
                            contractsforUser[i].actualBought = amountBought;
                            amountBought = 0;
                            fulfilled = false;
                        }
                    } else if (contractsforUser[i].sellId == _userId) {
                        if (amountSold >= contractsforUser[i].quantity) {
                            contractsforUser[i].actualSold = contractsforUser[i].
                                 quantity;
                            amountSold -= contractsforUser[i].quantity;
                            fulfilled = true;
                        }else{
                            contractsforUser[i].actualSold = amountSold;
                            amountSold = 0;
                            fulfilled = false;
                        }
                    }
                }
        return (fulfilled, contractsforUser);
    }


    function addContract(uint _buyId, uint _sellId, int _quantity, int _price)
         public isWhitelisted(msg.sender){
        if (auctions[currentAuction].finished){
            auctions[currentAuction].contracts.push(Contract(_buyId,_sellId,
                _quantity,_price,0,0));

        }
    }
    function addMultContract(Contract[] memory _contracts) public isWhitelisted(msg
         .sender){
        if (auctions[currentAuction].finished){
            for (uint i=0;i<_contracts.length;i++){
                auctions[currentAuction].contracts.push(_contracts[i]);
            }
        }
    }
    function getContracts(uint _auctionId) public view returns (Contract[] memory){
        return auctions[_auctionId].contracts;
    }
    function getContractsForUser(uint _auctionId, uint _user) public view returns (
         Contract[] memory){
        Contract[] memory contracts2 = new Contract[](auctions[_auctionId].
             contracts.length);
        for (uint i=0;i<auctions[_auctionId].contracts.length;i++){
            if (auctions[_auctionId].contracts[i].buyId == _user || auctions[
                _auctionId].contracts[i].sellId == _user){
                contracts2[i] = (auctions[_auctionId].contracts[i]);
            }
        }
        return contracts2;
    }

    function getClearing(uint _auctionId) public view returns (Clearing memory){
        return auctions[_auctionId].clearing;
    }


function quickSortDescending(Bid[] storage arr, int left, int right) internal {
        int i = left;
        int j = right;
        uint pivotIndex = uint(left + (right - left) / 2);
        int pivot = arr[pivotIndex].price;
        while (i <= j) {
            while (arr[uint(i)].price > pivot) i++;
            while (arr[uint(j)].price < pivot) j--;
            if (i <= j) {
                Bid memory a = arr[uint(i)];
                arr[uint(i)] = arr[uint(j)];
                arr[uint(j)] = a;
                i++;
                j--;
            }
        }
        if (left < j)
            quickSortDescending(arr, left, j);
        if (i < right)
            quickSortDescending(arr, i, right);
    }

    function quickSortAscending(Bid[] storage arr, int left, int right) internal {
        int i = left;
        int j = right;
        uint pivotIndex = uint(left + (right - left) / 2);
        int pivot = arr[pivotIndex].price;
        while (i <= j) {
            while (arr[uint(i)].price < pivot) i++;
            while (arr[uint(j)].price > pivot) j--;
            if (i <= j) {
                Bid memory a = arr[uint(i)];
                arr[uint(i)] = arr[uint(j)];
                arr[uint(j)] = a;
                i++;
                j--;
            }
        }
        if (left < j)
            quickSortAscending(arr, left, j);
        if (i < right)
            quickSortAscending(arr, i, right);
    }
        function getPriceCap() pure private returns(int){
        return 9999;
    }
    function computeClearing() public{
            int demand_quantity = 0;
            int supply_quantity = 0;
            int buy_quantity = 0;
            int remaining_sell = 0;
            int remaining_buy = 0;
            uint i = 0;
            uint j = 0;
            Clearing storage clearing = auctions[currentAuction].clearing;
            Bid[] storage bids = auctions[currentAuction].bids;
            Bid[] storage offers = auctions[currentAuction].offers;

            //sort arrays, consumer’s bid descending, producer’s ascending
            if (bids.length != 0){
                    quickSortDescending(bids, 0, int(bids.length - 1));
            }
            if (offers.length != 0){
                    quickSortAscending(offers, 0, int(offers.length - 1));
            }
            if(bids.length > 0 && offers.length > 0){

                Bid memory buy = bids[i];
                Bid memory sell = offers[j];
                remaining_sell = sell.quantity;
                remaining_buy = buy.quantity;
                while(i<bids.length && j<offers.length && buy.price>=sell.price){
                    if (remaining_buy > remaining_sell){
                        remaining_buy -= remaining_sell;
                        auctions[currentAuction].contracts.push(Contract(buy.user,
                             sell.user, remaining_sell, buy.price,0,0));
                        ++j;

                        if(j < offers.length){
                            sell= offers[j];
                            remaining_sell = sell.quantity;
                        }
                    }
                    else if (remaining_buy < remaining_sell){
                        remaining_sell -= remaining_buy;
                        auctions[currentAuction].contracts.push(Contract(buy.user,
                             sell.user, remaining_buy, sell.price,0,0));

                        i++;

                        if(i < bids.length){
                            buy = bids[i];
                            remaining_buy = buy.quantity;
                        }
                    }
                    else{
                        clearing.clearingQuantity = buy_quantity;
                        auctions[currentAuction].contracts.push(Contract(buy.user,
                             sell.user, buy.quantity, buy.price,0,0));

                        i++;
                        j++;

                        if(i < bids.length){
                            buy = bids[i];
                            remaining_buy = buy.quantity;

                        }

                        if(j < offers.length){
                           sell= offers[j];
                           remaining_sell = sell.quantity;
                        }

                    }

                }
                clearing.clearingQuantity = (demand_quantity > supply_quantity) ?
                     demand_quantity: supply_quantity ;
                if (i == bids.length || j == offers.length) {
                    clearing.clearingPrice = (i == bids.length) ? offers[j].price :
                         bids[i].price;
                } else {
                    clearing.clearingPrice = (bids[i].price > offers[j].price) ?
                         bids[i].price : offers[j].price;
                }

            }
            for (uint k=0;k<auctions[currentAuction].contracts.length;k++){
                auctions[currentAuction].contracts[k].price = clearing.
                     clearingPrice;
            }

    }


}
```

Code listing A.1: Contract.sol