Utilizing Blockchain Technology for Settlement in a Microgrid

Marit Schei Tundal

Master of Science in Cybernetics and Robotics
Submission date: June 2018
Supervisor: Geir Mathisen, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Utilizing Blockchain Technology for Settlement in a Microgrid

Candidate: Marit Schei Tundal

Supervisor: Geir Mathisen

Problem Description:

In a limited electrical system (a microgrid), such as an island community, there will be one or more energy producers (e.g. diesel generators), some consumers and some, zero or more, prosumers (consumers who also produce energy, e.g. by solar panel or wind turbines). In addition, there may be energy storages such as batteries.

The prosumer may at times produce more energy than they themselves use at the moment and sell this surplus energy to a neighbor who needs it. In an advanced solution, one can imagine that the prosumers have a separate battery that they can choose to charge instead of selling the energy. There are three sources of energy for the consumer in the described limited energy system; the pure manufacturers (e.g. diesel engines), the batteries (which sometimes buy energy) and the prosumers.

In the system, there are also consumers, who buy energy. The price of delivered energy may vary from source to source and over time. An optimization of energy based on different criteria may be interesting, but this will not be considered. In order to settle the value of the energy flowing in the system, one wants to look at the use of blockchain technology.

The tasks will be:

- Investigate how blockchain technology is used and can be used in microgrids.
- Suggest a blockchain-like system that can calculate the value of the energy flowing. This system may have a user interface where offer and demand are displayed.
- Implement the proposed system and test it using simulated data.

Preface

This paper is submitted on partial fulfillment of the requirements for the Master of Science degree at the Norwegian University of Science and Technology (NTNU).

The problem description for this thesis was given by supervisor Geir Mathisen, who has also provided helpful guidance in the system development, especially regarding understanding the problem description and development of functional specifications.

The system was implemented and tested on a standard PC, using Python. Several Python libraries were used in the implementation. These were *Twisted, Flask, ECDSA, plotly,* and *datetime*, as well as the Python standard libraries *os, csv, hashlib, uuid, random, pickle, json, time,* and *struct.* The implementation of the consensus model and the blockchain are based on findings from the background chapter.

Abstract

Blockchains have been embraced by many industries where transactions, either financial or non-financial, are involved. Furthermore, blockchains have been proven to support transactions, without the need for a third-party or other middlemen. Given the distributed properties of blockchains, they are very suitable as the underlying foundation of peer-to-peer applications. Among the peer-to-peer applications that could be improved with the support of blockchains, is the microgrid. Microgrids are finite energy systems where peers may purchase energy among themselves.

This thesis investigates the blockchains ability to work as the underlying technology to support settlement in a microgrid. The blockchain is essentially a distributed, immutable ledger. However, there exists many versions of this ledger based on implementation features such as governance and consensus etc. A thorough background study of blockchains, and the differences in the blockchain features are presented. Furthermore, existing systems using blockchains for settlement in peer-to-peer energy systems are presented. Based on the findings in the background study and related work, a new blockchain as the technology for supporting these transactions is proposed and implemented. Results from the tests show that the proof-of-concept blockchain developed in this thesis works well as the underlying technology for settling energy flow in a microgrid. However, further development and testing is required for a real system.

Sammendrag

Blokkjeder har blitt omfavnet av mange næringer der transaksjoner, enten finansielle eller ikke-finansielle, er involvert. Videre har blokkjeder vist seg å støtte transaksjoner hvor det ikke er behov for tredjepart eller andre mellommenn. Gitt de distribuerte egenskapene i blokkjeder, er de svært egnet som det underliggende grunnlaget for andre peer-to-peer-applikasjoner. Blant peer-to-peer-applikasjonene som kan forbedres med støtte fra blokkjeder, er mikrogrid. Mikrogrid er lukkede energisystemer der deltagere kan kjøpe strøm seg imellom.

I denne oppgaven undersøkes blokkjedenes evne til å fungere som den underliggende teknologien for å støtte oppgjør i et mikrogrid. En blokkjede er i hovedsak en distribuert, uforanderlig regnskapsbok. Det finnes imidlertid mange versjoner av denne regnskapsboken basert på forskjellige mekanismer som styring og konsensus etc. Et grundig bakgrunnsstudie av blokkjeder, og forskjellene i mekanismene blir presentert. Videre presenteres eksisterende systemer som bruker blokkjeder for oppgjør i peer-to-peer-energisystemer. Basert på funnene i bakgrunnsstudiet og relatert arbeid, foreslås og implementeres en ny blokkjede som den underliggende teknologien for å støtte transaksjonene. Resultatene fra testene viser at den foreslåtte blokkjeden, som er utviklet i denne oppgaven, virker som den underliggende teknologien for å avgjøre energiflyt i et mikrogrid. Likevel er det nødvending med videreutvikling og testing før det kan brukes i et ekte system.

Abbreviations

API	=	Application Programming Interface
ASIC	=	Application Specific Integrated Circuits
BFT	=	Byzantine Fault-Tolerance
BGP	=	Byzantine Generals Problem
CSV	=	Comma Separated Value
DAO	=	Decentralized Autonomous Organization
ECDSA	=	Elliptical Curve Digital Signature Algorithm
EVM	=	Ethereum Virtual Machine
NOK	=	Norwegian Kroner
P2P	=	Peer-to-Peer
PBFT	=	Practical Byzantine Fault Tolerance
PoC	=	Proof-of-Concept
PoS	=	Proof of Stake
PoW	=	Proof of Work
RPC	=	Remote Procedure Call
UUID	=	Universally Unique Identifier

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

In 2008, after years of granting loans to "sub-prime" clients who struggled to repay their mortgage, the investment bank Lehman Brothers, filed for bankruptcy [1]. The events that followed launched a global financial crisis where stock markets dropped, and unemployment rates increased worldwide. The banks who initially caused the crises, however, were bailed out, using tax payer money. People no longer felt they could trust bankers or investment managers. As a response to this, Satoshi Nakamoto outlined a trustless peer-to-peer (P2P) electronic cash system in the Bitcoin whitepaper.

When the bitcoin system became a reality at the start of 2009, the very first block contained the message "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks". Furthermore, Nakamoto posted the following on an internet P2P forum [2] in 2009:

"*The root problem with conventional currency is all the trust that's required to make it work. The central bank must be trusted not to debase the currency, but the history of fiat currencies*[1] *is full of breaches of that trust. Banks must be trusted to hold our money and transfer it electronically, but they lend it out in waves of credit bubbles with barely a fraction in reserve. We have to trust them with our privacy, trust them not to let identity thieves drain our accounts. Their massive overhead costs make micropayments impossible.*"

The distributed system was backed by mathematics and cryptography, instead of the traditionally trusted middlemen and third-parties. The Bitcoin technology

[1]Fiat currency is currency backed and issued by the government, such as U.S. dollars or Norwegian kroner

removed the need for trust in transactions, due to the distributed, decentralized ledger where all transactions are stored. Since all transactions are stored across all nodes in the network, there is no single point of failure. Unlike traditional databases, this distributed system cannot be hacked, and once data is stored in the ledger, it cannot be altered.

Cryptocurrencies were not invented by Nakamoto, they existed already in the 1980's, but never saw any real usages as they had some problems. What set Bitcoin aside, was the solution to the *double spend problem*[2]. Nakamoto solved this by timestamping all transactions and storing them in a immutable, decentralized ledger - the blockchain.

Bitcoin and *cryptocurrency* have become household words, with nearly daily appearances in media. Although many people still use the terms *Bitcoin* and *blockchain* interchangeably, they are not the same. Bitcoin is, simply put, the first application created using blockchain technology. Parallels can be drawn to the early days of the Internet, where TCP/IP (Transmission Control Protocol/Internet Protocol) became the technology that allowed e-mail to become a reality [3], just like the blockchain technology enables Bitcoin.

Since the introduction of blockchains, there have been massive developments in the field with the creation of many new applications, both financial and non-financial. With usage ranging from bank-to-bank transfer, to voting, recording landownership, and sale and licensing of intellectual properties, blockchains could impact many industries.

The electricity market is among the industries who have adopted the blockchain as a method to improve transaction. In 2014, a method to trade renewable energy through virtual currency was proposed with the NRGcoin [4]. The first successful energy transaction to be executed in a blockchain occurred with the Brooklyn Microgrid project in 2016 [5]. In spite of being a relatively new technology, blockchains in the electricity market have shown great potential. Especially in a P2P system, such as microgrids, the potential for blockchains are promising. Since there is no central utility that distributes the electricity, there does not need to be a central authority controlling settlements in the system.

1.2 Limitations

The system mainly consists of two parts: implementation of a blockchain for storing data in a distributed network; and an application for settlement[3] of energy transactions in a microgrid. The blockchain is the back-end service enabling the functionality of the application, and the main focus of this thesis. In a complete system for settling transactions in an electricity system, there are several aspects

[2]If person A has an asset and sends this to person B and C simultaneously, which transaction should be valid.

[3]Resolving payment.

that ought to be considered. Among these are privacy, however, the system implementation in this thesis is focused around the technical design and implementation. Therefore, privacy is not part of the implementation, but will be further discussed in chapter 9. Another important aspect that should be considered in a complete system is the optimization of the settlement algorithm. This is a very complex task and is not part of the scope of this thesis. A well-functioning user-interface is required for the proposed system to work. This task is only outlined in the design and implementation chapters, as a full functional user-interface is deemed too time consuming to develop, and the main focus of the implementation is centered around the blockchain. The task of creating a fully functional smart contract Application Programming Interface (API) is also deemed as too complex a task to be implemented in the limited time of this thesis. However, this process is outlined in section 5.2.2 and further discussed in chapter 9.

1.3 Contribution

Contributions in this thesis are summarized as:

1. Presentation of relevant background theory relating to the system.
2. Outline existing implementations of blockchains used in peer-to-peer electricity trading.
3. Propose a new blockchain-like system for storing electricity transactions.
4. Propose a simple method for settling electricity transactions in the system.

1.4 Thesis Outline

Chapter 2 presents a short description of microgrids and a theoretical background of blockchains, starting with an introduction to blockchains. Furthermore, taxonomy and mechanisms such as consensus are explained. Existing blockchain applications are also discussed. The background research is quite extensive, as this is a fairly new technology. Related work on projects using blockchains in energy transactions is presented in chapter 3.

The functional specifications of the system to be implemented are presented in chapter 4. A system design is illustrated in chapter 5, while the implementation of the system is described in chapter 6. Testing and results are presented in chapters 7 and 8, respectively, while discussion and conclusion of the thesis will be presented in chapters 9 and 10, respectively.

Chapter 2

Background

This chapter contains relevant background theory on blockchains. The chapter starts off with a very basic introduction to microgrids.

2.1 Microgrids

A microgrid is a local power grid that may or may not be connected to the main grid. The microgrid can either work as an extension of the main power grid, or operate in "island mode" and function autonomously. Hayden [6] described microgrids as:

"*A Microgrid is a group of interconnected loads and distributed energy resources within clearly defined electrical boundaries that acts as a single controllable entity with respect to the grid*".

Microgrids usually include renewable energy sources, such as wind generators, small hydro power generators, and photovoltaic solar panels, that are locally grouped together. Microgrids may also include generators and batteries to ensure reliability in the network [7]. By placing power generation and usage closer together, efficiency will be increased and transmission losses reduced [8].

A key benefit of the microgrid, is the ability of being self-sufficient in case of emergencies that cause blackouts. By disconnecting from the grid, the locale area powered by the microgrid can operate for days without any connection to the main grid. This happened, for instance, in New York during hurricane Sandy in 2012 [9]. The city was without power for several days, but hospitals and other key facilities could operate due to microgrids.

Places that are not connected to the main grid have very expensive solutions for receiving power. This can typically be rural places or islands. These places may also benefit from microgrids. The ability to operate in island-mode will provide better electricity services at a lower cost than e.g. importing diesel [10].

2.1.1 Electrical Transmission

In microgrids, just like in the traditional power grid, there is no way of telling where the consumed power is actually generated. Once a power plant or another power producer put electricity in to the grid, the electrons from all sources are mixed together before they are transferred to consumers [11]. The electricity sources are indistinguishable. In a microgrid, this means that even if producer A has a contract with consumer B to deliver electricity, consumer B is not necessarily using the electricity that producer B has generated. If producer C is also generating electricity to the gird, there is a chance that consumer B is actually consuming the electricity from producer C. Electrons in the grid always flow from source (generators) to sink (loads), where electricity is being consumed [12].

2.2 Introduction to Blockchain

A blockchain is a decentralized ledger, distributed over a network of nodes. Transactions in the network are stored in blocks which are linked together as a chain - creating the blockchain. It was first introduced by Satoshi Nakamoto with the Bitcoin application [13].

The blockchain is primarily used to transfer assets between users. In contrast to previous transaction methods, there is no need for trust between the participants of the transaction, nor the need of a trusted third-party institution, like a bank or government. The trust lies in the system and the mathematical functions behind it, and not the individual participants [14]. Due to the decentralization of the network, there is no single-point-of-failure, as the ledger is duplicated on every node of the network.

The following description of the blockchain technology is based on the Bitcoin blockchain proposed in Nakamotos whitepaper from 2008 [13]. Many of the features from the Bitcoin blockchain hold true for most blockchains. However, there are many variations of blockchain implementations, due to public/private configurations, consensus models etc. These differences will be described later in the chapter.

When a node transfers an asset to another node, it creates a transaction. The transaction contains one or more inputs, one or more outputs, and a digital signature. In Bitcoin, an input is a previously received and unspent output, containing a certain amount of bitcoins. Multiple inputs can be added together to create a larger transaction. The inputs are used completely, so if the amount does not add up to the wanted output, multiple outputs can be used so that the change is sent back to the owner. This is because transactions must reference an output, and each output can only be referenced once. Thus, the exact ownership of every asset can be pinpointed to a user in the network, based on the previous transactions. In order for a transaction to occur, it must be proved that the asset has not been previously spent by the current owner. Thus, there is no way to double spend assets.

Transactions are validated through digital signatures, proving the authenticity of a message, by using public and private keys. Each node can obtain a random private key, and a corresponding public key derived from the private key. If node A wants to send a bitcoin to node B, it must use B's public key, which acts as the address to B's Bitcoin wallet. Node A creates a digital signature of the transaction with its private key. The validity of the transaction can then be confirmed using A's public key. When the transaction is confirmed, the bitcoin can only be spent by the person in possession of the private key corresponding to B's public key. This transaction is illustrated in figure 2.1. The use of digital signatures ensures that transactions can traverse the network without being altered, as even the smallest change will cause a signature to no longer be valid.

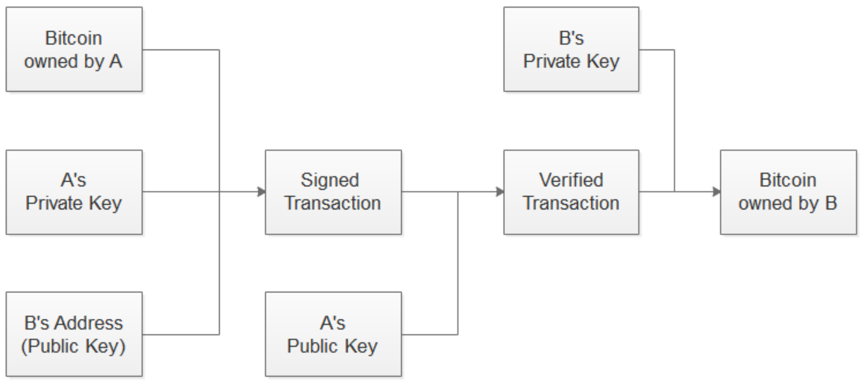


Figure 2.1: Digital signature in Bitcoin transaction.

A miner creates new blocks by storing multiple pending and unconfirmed transactions together. The block consists of the transactions, as well as the block header. The block header usually contains: a reference to the previous block in the chain, namely the hash of the previous block; a timestamp for when the new block is created; and a nonce[1] and a target used to calculate the hash of the new block.

When the hash of the new block is found, the block is broadcasted to the rest of the network. Every node on the network validates the block, before they start working on the next block. Validation is an easy task, once the hash is found. If several

[1]A variable number used to find the target hash.

miners find the solution to the next block at the exact same time, a fork is created. This sequence is illustrated in figure 2.2

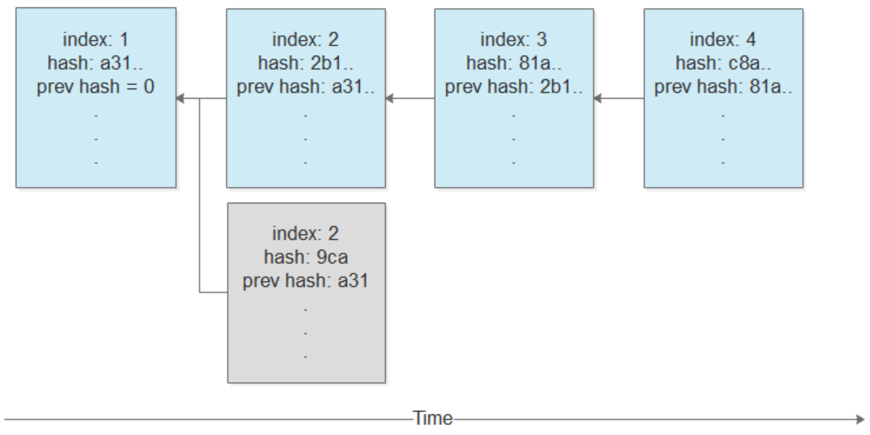


Figure 2.2: Connection between blocks.

A fork means that there exists two (or more) different versions of the blockchain in the network. Due to network latencies, nodes in the network will receive different broadcasted messages at different times. Therefore, nodes might have different versions of the blockchain. The miners start working on the next block based on the one they received first, while storing the other version in case that fork becomes longer [13]. The fork is resolved when one side of the fork becomes longer than the other(s), which will most likely happen when the next block is created. The side with the longest chain always wins because it is backed by the most work. All the nodes in the network immediately look to the longest chain as the valid version of the blockchain. The transactions in the discarded block go back into the pool of the pending transactions, waiting to be mined in a block. Once a transaction is part of the blockchain, it is irreversible. Due to the hash functions, even the slightest alteration in a transaction will be detected, and the transaction will be deemed as non-valid. Thus, a transaction cannot be changed or reverted once it is part of the blockchain.

There is, however, some need for caution. As previously mentioned, transactions in discarded blocks go back to being unconfirmed. If the other side of the fork contains a transaction attempting to double spend the asset, the initial transaction might no longer be valid. Say, for instance node A sends two bitcoins to node B, while also sending the same two bitcoins to itself, only one of these transactions can be valid. Once a transaction is stored in the blockchain, the two bitcoins are referenced as a new unspent output, and the other transaction is no longer valid. This is intentional behavior in the blockchain, as it solves the problem of double spending which is usually what a trusted third-party, e.g. a bank, is needed for. The problem occurs when there is a fork. If node B receives the bitcoins as a payment for some goods, and ships the goods once the transaction is part of the blockchain, while node A is simultaneously working on another fork in the blockchain containing the transaction sending the two bitcoins to itself, node B will lose the payment for the goods if the fork node A is working on becomes longer. For this reason, it is a good policy to wait until the block containing the transactions has multiple successors before asserting the transaction as final.

2.3 Blockchain Taxonomy

This section details the taxonomy of blockchains, in particular how they are governed and the level of authorization needed in the different classifications. Blockchains are usually divided into three different categories, depending on the way they are governed [15, 16, 17].

- Public blockchains, like Bitcoin, are blockchains where anyone is free to participate, and is completely decentralized.
- Private blockchains are more centralized with e.g. a single organization controlling the blockchain.
- A consortium blockchain is a hybrid between public and private, where participants must be authorized, but there is no single, central authority controlling it. It is also described as "*[...] a traditional centralized system with a degree of cryptographic auditability attached*" [15].

Authorization in the blockchain depend on whether they are permissioned or permissionless. Permissionless blockchains are open to anyone, while permissioned blockchains require approval from the owner or existing members to join. Public blockchains are categorized as permissionless, while private and consortium blockchains fall under the category of permissioned.

The characteristics of public, private and consortium blockchains are summarized in table 2.1.

In conclusion, the type of blockchain most beneficial for each application will vary from industry to industry and from application to application. Public blockchains are most beneficial for individual users, as there is no need for trusted third-parties or trust between users. Private and consortium blockchains are more beneficial to organizations or companies, as they lower cost and increase speed of transactions.

Table 2.1: Comparison between public, private, and consortium blockchains.

	Public	**Private**	**Consortium**
Consensus	All miners	One organization	Some nodes
Authorization	Permissionless	Permissioned	Permissioned
Performance	Slow	Very fast	Fast
Security	Nearly impossible to alter	Could be altered	Could be altered
Centralization	No	Yes	Partially
Computational expensiveness	Costly	Cheap	Depends on consensus
Area of use	Crypto-currency	One organization	Several organization with some trust
Anonymity	Yes	No	No

2.4 Consensus Models

Consensus models are important in distributed systems, in order to reach distributed agreement and maintain a correct and concise version of the shared ledger between the nodes. There are two different types of fault-tolerance in consensus mechanisms [18]. Fail-stop fault-tolerance means that the system is able to perform correct behavior if it is able to detect nodes that have failed. Byzantine fault-tolerance (BFT) means that the system performs correct behavior even if there are malicious nodes that intentionally try to cause failures in the system. BFT protocols are designed to be a solution to the Byzantine Generals problem (BGP) [19]. Roughly explained, the BGP is a compliance problem where generals in the Byzantine army must agree upon whether to attack or retreat, using only messages, even if there is a traitor among them. Similarly, blockchain nodes must agree on the validity of a block using only messages, even if there is a Byzantine node present.

2.4.1 Proof of Work

Proof of Work (PoW) is the consensus model presented by Nakamoto in the Bitcoin white paper [13], and is a commonly used consensus protocol in blockchains. In the PoW consensus model, nodes are required to put a certain amount of work into creating new blocks to avoid DDoS-attacks and to make it harder for malicious nodes to hijack the blockchain [17]. The required work consists of solving a computationally expensive and complex cryptographic problem. Each new block has a unique hash based identifier. The problem consists of combining this hash with a nonce, to find a new hash that is less than a given target (e.g. a 256 bit hash containing x significant zeros) [20]. While the proof is somewhat difficult to find, it is very fast and easy to verify by the rest of the network.

By lowering the target value, the problem can be adjusted to be more difficult. The amount of work required is exponential to the number of significant zero bits. This implies that untrustworthy peers wanting to modify past blocks have to work much harder than peers adding new blocks, due to the work required to redo already mined blocks. This can be seen in figure 2.3

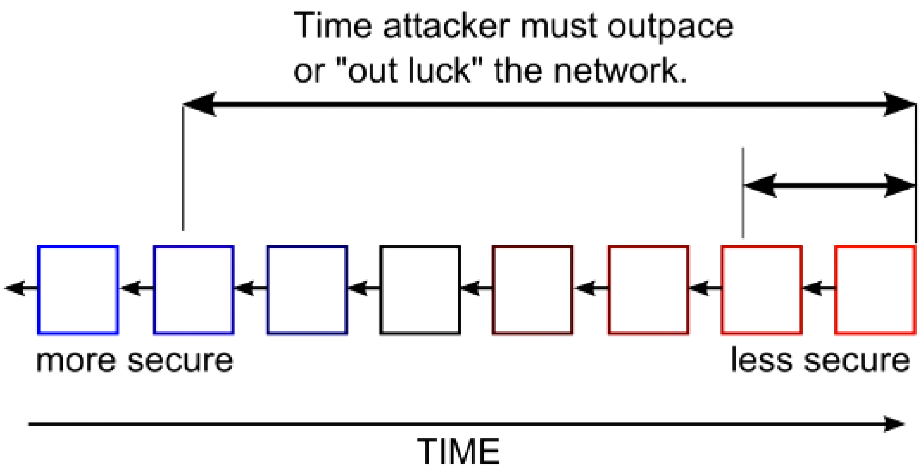


Figure 2.3: Transactions in blocks are in less danger of being altered as more time passes [21]

Nodes working on finding these proofs are called miners, and receive some form of reward for finding the correct proofs (e.g. a certain amount of the cryptocurrency or transaction fees). Special Application Specific Integrated Circuits (ASIC) computers are created for mining. These computers solve the hash problems at higher rates and with less energy consumption than normal computers [22]. By letting anyone participate as a miner it is ensured that the network has complete decentralization.

As described by Nakamoto, PoW is also a method for ensuring that a majority always controls the blockchain [13]: "*If the majority were based on one-IP-address-one-vote, it could be subverted by anyone able to allocate many IPs. Proof-of-work is essentially one-CPU-one-vote*". The longest chain reflects the most proof-of-work and, thus represented by the majority in the network.

Some of the major problems related to PoW come from the latency and vast amounts of energy required to mine new blocks. As of May 2018, the Bitcoin blockchain consumes more energy than the entire country of Switzerland [23], as seen in figure 2.4.

Nakamoto shows that the probability of an attacker altering a block decreases as more blocks are added on top of it [13]. However, the probability never reaches zero. An example of this is the 51 % attack, where a group wishing to attack the blockchain gains 51 % of the computation power, and thus is in control of the blockchain. By obtaining control of the majority computing power, the group can generate the longest, and thereby valid, blockchain.

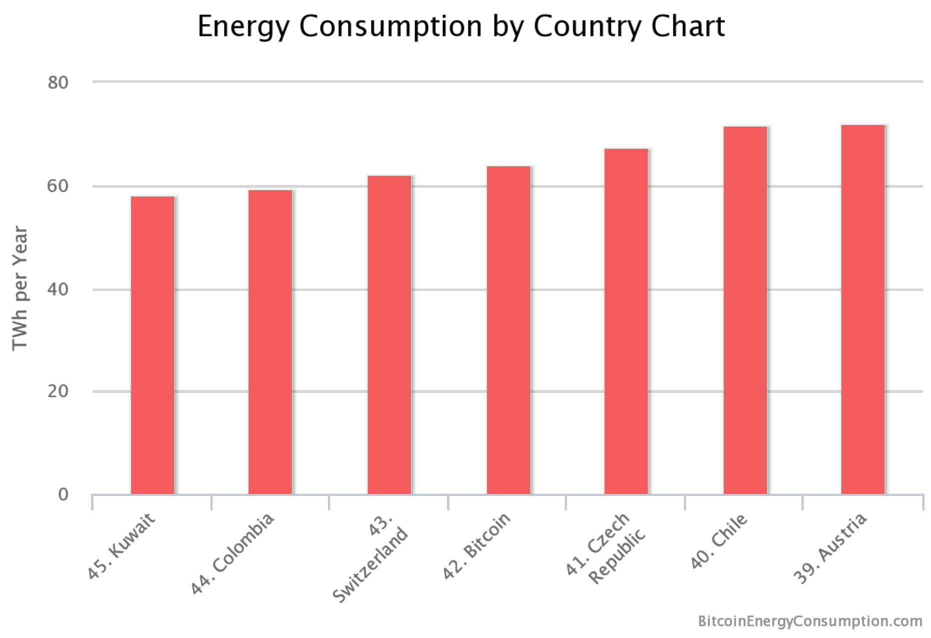


Figure 2.4: Bitcoins energy consumption index relative to countries in the world [23]

PoW ensures that the blockchain eventually reaches consensus. Due to the possibilities of forks in the chain, it might take some time for the network to agree on the final version of the ledger. This results in slow transaction time with only 7 transactions per second, as compared to VISAs throughput of 10 000 transactions per second [24].

2.4.2 Proof of Stake

Proof of Stake (PoS) was created to solve the energy problem of PoW [24, 17]. Nodes are chosen to participate in the consensus process in a deterministic manner, based on the amount of stake they have in the system. This means that if a node, or stakeholder has 10 times more assets in the network than another node, it is 10 times more likely [25] to be chosen to forge new blocks and hence receive a reward. The reward in PoS comes from transaction fees, as there are no new coins to be mined, or created, since all coins exists from the start of the blockchain.

The creator of a new block puts his or her own coins at stake. This means that if they produce a block containing fraudulent transactions, they lose their stake. After putting up the stake, the node becomes part of the validation process. Due to their stake in the block, they are incentivized to only validate the correct transactions. The problem occurs when a validator has very little, or nothing, at stake. It is more economical profitable to vote on multiple blocks, supporting multiple chains in order to maximize expected rewards [24, 18]. This problem is known as the nothing-at-stake problem. Ethereum, which is currently using PoW but plan to change to PoS in a future release, proposed a solution to the nothing-at-stake problem by penalizing validators who vote for the wrong block [25].

A major problem with this model is that the rich get richer, due to the selection process favoring nodes with more stake. This, in turn, implies that the network will be more centralized over time. However, it can be argued [26] that PoW favors the rich even more. Due to electricity cost, and costly hardware required for mining, this could be more expensive than buying crypto-currency in the blockchain, in order to be more likely to participate in the validation process [27].

2.4.3 Raft

Raft is a partially asynchronous, leader based consensus algorithm for managing replicated logs across a distributed system [28]. Raft was designed to be a more understandable alternative to the Paxos consensus protocol [29]. The Raft algorithm was first proposed by Onargo et al [28]. Following is a description of the algorithm.

The Raft algorithm consists of three main parts: *leader election*, *log replication*, and *safety*. Any node participating in the consensus process is in one of three states: leader, follower, or candidate. The interchanges between these three states can be seen in figure 2.5.

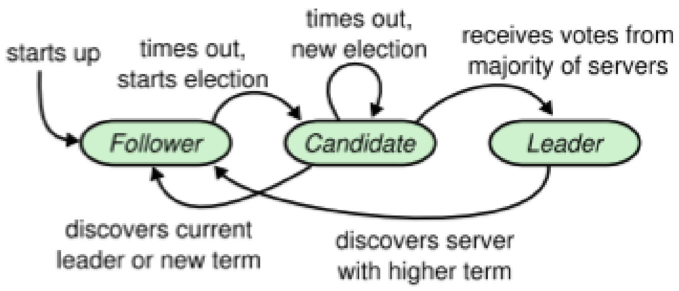

starts up
times out, starts election
times out, new election
receives votes from majority of servers
Follower
Candidate
Leader
discovers current leader or new term
discovers server with higher term

Figure 2.5: Server states. [28]

In Raft, time is divided into terms. Each term is of arbitrary length, depending on the operation of the leader. Each term starts with a *leader election.* If a follower node has not heard anything from a leader after a certain amount of time it becomes a candidate, and request votes from other nodes in the network. There are three possible outcomes for a candidate:

1. The candidate receives a majority of votes and becomes leader.

2. The candidate receives a message from another node claiming to be the leader. The candidate then steps down and becomes a follower.
3. If the candidate does not receive a majority, e.g. due to a split vote, it will start a new term and a new election.

The algorithm guarantees that the following properties are true at all times, thus ensuring consensus [28]:

- "**Election Safety:** *at most one leader can be elected in a given term*"
- "**Leader Append-Only:** *a leader never overwrites or deletes entries in its log; it only appends new entries*"
- "**Log Matching:** *if two logs contain an entry with the same index and term, then the logs are identical in all entries up through the given index*"
- "**Leader Completeness:** *if a log entry is committed in a given term, then that entry will be present in the logs of the leaders for all higher-numbered terms*"
- "**State Machine Safety:** *if a server has applied a log entry at a given index to its state machine, no other server will ever apply a different log entry for the same index*"

When the system operates under normal conditions, there is exactly one leader and all other nodes are followers. The followers are passive, and only respond to requests from leaders or candidates through Remote Procedure Calls (RPC). Each *log entry* consists of an index and a term, as well as some data. If two entries in different logs have the same index and term, all entries up to that point are guaranteed to be identical. A leader cannot overwrite any committed log entries, only append new entries.

Safety is ensured in Raft by putting restrictions on which nodes can become leaders. A candidate must include all previous committed entries in its log, in order to become a leader. The leader also ensures that any followers missing entries are brought up to date.

Raft is not a BFT consensus protocol, it is only fail-stop fault-tolerant. All nodes participating in the consensus are assumed to be trusted, meaning that there are no Byzantine faulty nodes.

2.4.4 Practical Byzantine Fault Tolerance

A Byzantine fault-tolerant algorithm is one that can tolerate Byzantine faults that usually occur in network systems with multiple nodes communicating. These types of faults include [30]:

- Messages lost due to network failures.
- Messages corrupted through malicious nodes.

- Messages compromised through man-in-the-middle attacks.
- Hardware or software failures.
- Messages from nodes without permission to join the network.
- Arbitrary behavior.

The Byzantine Generals problem was first proposed and solved by Lamport et al. [19]. However, the solution presented in the paper is expensive regarding time and message overhead, thus making it impractical to use in real systems. A solution to this problem was proposed by Castro et.al [30] with their new *Practical Byzantine Fault Tolerance* (PBFT) algorithm.

The PBFT algorithm states that the system is able to make progress with f faulty nodes, if the total number of nodes n, is

$$n = 3f + 1$$

Out of the n nodes in the consensus model, one is the primary, which receives requests from the client, and the rest of the nodes are replicas. After receiving a request, the primary initiates a three-phase protocol, (pre-prepare, prepare, and commit) to ensure consensus. Results from all replicas are sent to the client, who in turn accepts the result if there are f+1 identical replies. In the pre-prepare phase, the request is given a unique sequence number, which the replicas agree on in the prepare phase. The commit phase establishes total order across views. The order of communication in the three-phase protocol can be seen in figure 2.6.

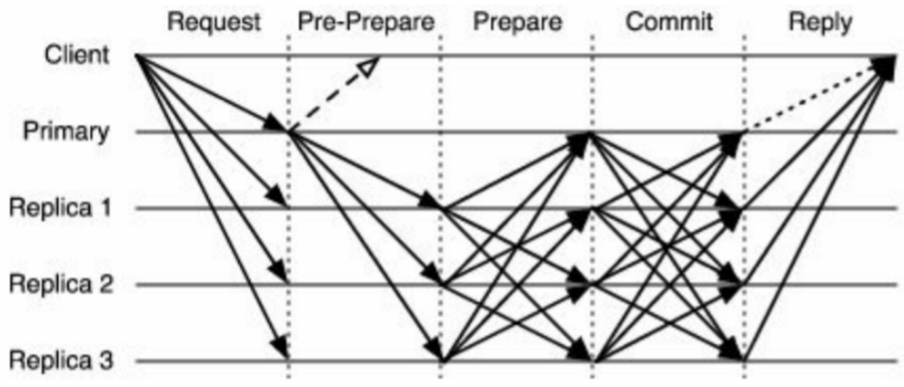


Figure 2.6: Three-phase protocol for consensus in PBFT [30].

2.4.5 Summary of Consensus Models

Table 2.2: Consensus models

Protocol	**PoW**	**PoS**	**Raft**	**PBFT**
Requires crypto	Yes	Yes	No	No
Leader	No	No	Yes	Yes
Pros	Decentralized	Energy efficient	Low Cost Energy efficient	High throughput Low cost
Cons	Slow throughput Energy inefficient	Nothing at stake	Not BFT	Semi-trusted
Implementations	Bitcoin Ethereum	Peercoin Ethereum		Hyperledger Tendermint

This is perhaps the most challenging aspect in order to create a blockchain. Different types of blockchains require different types of consensus models. At this point, no single best consensus model exists. Limitations and restrictions are present in regard to power consumption, scaling, security, and transaction speed.

2.5 Blockchains and Their Applications

Numerous blockchains have been created since the Bitcoin blockchain was developed. Many blockchains, like Bitcoin, are based around crypto-currency. However, there are several blockchains with other purposes than being purely a crypto-currency, such as the Ethereum blockchain. Some of these blockchains will be discussed in this section.

2.5.1 Bitcoin

How Bitcoin works was roughly described in section 2.2. This section will discuss what Bitcoin is used for, and in what applications it is beneficial. Bitcoin is first and foremost a crypto-currency blockchain, so the applications are financial.

The Silk Road drug market [31], albeit illegal, is perhaps the field where Bitcoin provided the most convenience for its users, before it was shut down by the FBI in 2013. By using Bitcoins as payment, the marketplace ensured anonymous drug trading. In its 2.5 years of operation, the dark web market had sales worth over 1 billion USD [32]. This emphasizes the obviously negative sides of Bitcoin, that the anonymity of the transactions makes it easier to finance illegal activities.

A major advantage of Bitcoin is the low costs of transferring money abroad [33]. Compared to traditional transfers between banks which are both costly (on average 7.45% [34]) and might take up to several days, Bitcoin has no extra transaction fees and new transactions are processed in minutes or hours, depending on the network traffic [35].

An example of an application built on top of the Bitcoin blockchain is Lighthouse [36]. Lighthouse is a crowdfunding application where users can make donations to projects, using bitcoins.

2.5.2 Ethereum

Ethereum is an open-source blockchain application platform [37] proposed by co-founder Vitalik Buterin [38] in 2013 in his white paper [39]. The decentralized platform runs smart contracts, which are unstoppable application transactions. Smart contracts are further described in section 2.6.

Inspired by Bitcoin, Buterin came up with a platform to expand the limited financial use cases available in the Bitcoin blockchain. Ethereum is a programmable blockchain. Protocols run on the Ethereum Virtual Machin (EVM) [40]. Applications are created by users in the Turing complete programming language[2], Solidity, and executed on the EVM. When an operation is executed on an EVM, it is executed simultaneously on all nodes in the network.

Applications on Ethereum can either be financial, non-financial or semi-financial. Like Bitcoin, Ethereum also has a native crypto currency, Ether. However, unlike Bitcoin, Ether has a purpose other than simply being a crypto-currency. In order to run applications on the Ethereum blockchain, a small fee, or gas price, is required. Gas is purchased with Ether. The use of gas ensures efficient and economical code, as each computational step in the code requires a certain amount of gas [39].

Whereas Bitcoin only allows users to interact with the blockchain through a set of pre-defined operations, Ethereum allows users to create applications with arbitrary complexity.

Several decentralized applications have been created on top of the Ethereum blockchain. Among these is the Brooklyn Microgrid, which will be further discussed in section 3.1. Other applications include Decentralized News Network [41], where factual news reports are rewarded with tokens and free of censorship; uport [42], an identity system for the decentralized web where users can store their identity and credentials on the Ethereum blockchain.

2.5.3 Hyperledger

The Hyperledger project consists of over 100 members, including companies such as IBM, Intel, Samsung, J.P. Morgan, American Express and Airbus [43]. It is an open-source platform for blockchains hosted by the Linux Foundation. It consists of multiple blockchains, including Hyperledger Fabric from IBM and Hyperledger Sawtooth from Intel.

[2]A Turing complete programming language is one that can solve any problem that a Turing machine can.

The Hyperledger blockchains differ from each other, but all have in common that they do not have any native crypto currency. The Linux foundation focuses on open industrial blockchain development:

"*Hyperledger is an open source collaborative effort created to advance cross-industry blockchain technologies. It is a global collaboration, hosted by The Linux Foundation, including leaders in finance, banking, Internet of Things, supply chains, manufacturing and Technology.*" [43]

The rest of this section is focused on Hyperledger Fabric. Fabric version 1.0 was released in July 2017, and the new version Fabric v1.1 was released in March 2018. The platform is a base for developing blockchain with modular architecture [44], and aims to be a plug-and-play solution for various applications. Among the modules is a PBFT consensus model. Through the smart contract system in chaincode, the application logic is determined.

As of yet, the Hyperledger projects are still fairly new and mostly limited to Proof-of-Concept (PoC) testing [45]. The finance sector is one of the industries where the Hyperledger is in the PoC stage. Some of the benefits of using a distributed ledger technology in the financial industry includes: "*streamlined settlement, improved liquidity, supply chain optimization, increased transparency, and new products/markets*" [45].

Following is a list of PoC projects that have been developed within the finance sector using Hyperledger technology:

- Bank-to-Bank transfers [46]
- eVoting [47]
- Peer-to-peer energy trading [48]
- Cross-border trade operations [49]

In 2016, transport company Maersk partnered with IBM to create a platform where blockchains are used in the cross-border shipping processes [50]. The goal is to provide a better and more efficient solution for tracking information and handling documentation in the shipping industry. The system is built using Hyperledger Fabric. Each shipment will have its own blockchain, containing only transactions and information relevant to that shipment. Through a user interface, participants can connect to the system and monitor the shipment. However, participants will only have access to transactions relevant to them, thus ensuring privacy in the blockchain.

2.5.4 Comparison of Commercial Blockchains

Table 2.3 summarizes the key features of the previously discussed blockchains:

Table 2.3: Comparison between blockchains [13, 40, 43]

	Bitcoin	**Ethereum**	**Hyperledger Fabric**
Founded	2009	2015	2017
Crypto currency	Bitcoin	Ether	Non
Consensus	PoW	PoW, PoS	PBFT
Permission	Permissionless	Permissionless	Permissioned
Public access	Public	Public and Private	Private
Smart contracts	Non	Smart Contracts	Chaincode
Language	C++	Go, C++, Python	Go, Java

2.6 Smart Contracts

The idea of smart contracts was first proposed by Nick Szabo in 1996, stating that *"Smart contracts combine protocols, user interfaces, and promises expressed via those interfaces, to formalize and secure relationships over public networks. This gives us new ways to formalize the digital relationships which are far more functional than their inanimate paper-based ancestors. Smart contracts reduce mental and computational transaction costs, imposed by either principals, third parties, or their tools [51]"*.

Smart contracts can be compared to vending machines regarding their ability to self-execute, given that the right terms are met. For instance, a vending machine will only provide the user with the goods, given that the purchased item is in stock and that the user has put in enough money. Just like with smart contracts, it is an event based mechanism that is executed without the need of a third party.

The Ethereum blockchain was built with smart contracts as the main objective. In this case, a smart contract is code which enable self-enforcing applications, through a pre-defined set of rules in the code. The smart contract code is stored in blocks which in turn becomes part of the blockchain. The smart contracts are used to establish rules between peers. Smart contracts add layers of information onto digital transactions being executed on a blockchain. It allows for more complex transactions compared to exchanging digital tokens for a product or service.

However, smart contracts are not without issues. The Decentralized Autonomous Organization (DAO), an entity based on Ethereum, implemented a crowd funding platform that raised 150 million USD. Due to shortcomings in the implementation of the contracts, hackers were able to withdraw 60 million USD from the funds [52]. This shows that smart contracts can only be as smart as the people coding them. Other issues include the difficulties to update or alter contracts once they are stored on the blockchain, due to their immutable properties. If a bug in the contract is detected, it is not easily patched.

In summary, smart contracts are just computer code that is stored on the blockchain, and runs when the right conditions occur. The key characteristics can be summarized as follows:

- Contracts consist of pre-defined terms and contract-parties.
- Contract execution is triggered by events, such as received transaction information.
- Terms of contract dictate where to move value when terms are met.
- Digital assets are automatically settled on the chain once a contract is executed.

Chapter 3

Related work

There already exist some examples of blockchains being used in the energy market.

Using blockchain in the energy market was first proposed by Mihaylov et al. [4]. NRGcoin was introduced as digital currency for buying and selling energy through the use of smart contracts. However, the transactions are not done in a P2P approach.

There are several companies emerging, that are developed for P2P energy trading. Three of these, all entering the market in 2016 or later, will be further discussed.

3.1 Brooklyn Microgrid

The Brooklyn Microgrid is a project owned by LO3, where some neighbors in Brooklyn, New York trade energy amongst themselves. [5]. This is done in a P2P manner, over the blockchain, thus removing the middleman. Residents who have installed solar panels on their roof tops, can sell any excess energy to their neighbors. The first transaction occurred in April 2016, and is [53, 54] the first ever energy transaction made on a blockchain.

The company started working with the public blockchain Ethereum [55, 54]. However, they quickly realized that this was not a feasible solution in terms of scalability, and decided to make their own, private blockchain.

The microgrid was developed by the Siemens Digital Grid Division, while the blockchain platform was developed by LO3 Energy. Each participant in the project has a meter installed. These meters communicate with each other and other smart devices. Furthermore, they measure the energy production and consumption. The key factor, however, is that these meters form the blockchain [55, 56]. Transactions from each meter is stored on the blockchain, while all meters contribute with the computational power necessary to run the blockchain and validate the transactions.

Mengelkamp et al. [57] presented a case study of the Brooklyn Microgrid. The project consists of both a virtual platform and a physical microgrid. The microgrid is built in parallel with the existing grid, and is able to uncouple from the traditional grid in case of power outages. The virtual platform is the technical infrastructure, and runs a private blockchain, whereas the virtual layer is where all the information of the network is transferred.

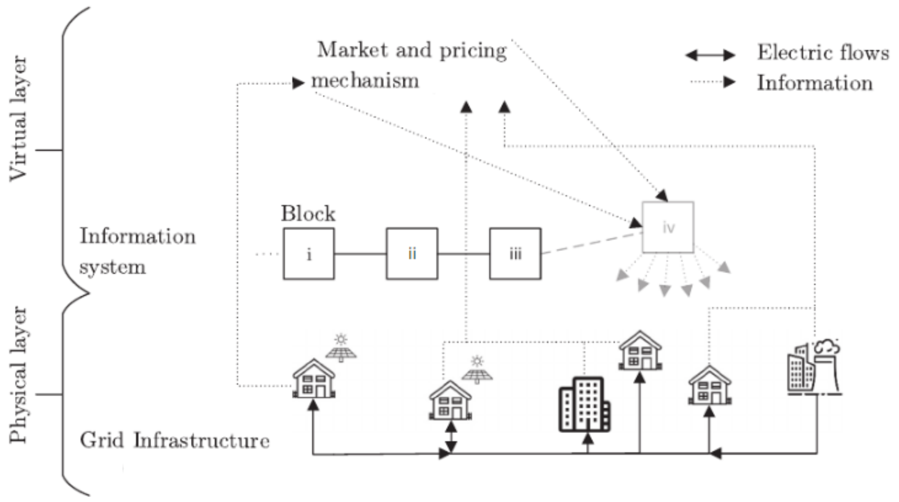


Figure 3.1: High-level topology of the Brooklyn Microgrid [57].

3.2 NAD Grid

NAD Grid is a platform for P2P energy transactions over the blockchain, created on the Hyperledger blockchain [48]. The company focuses on reducing carbon footprint through their Selective Sourcing technology [58].

The software platform is built upon the physical grid, and they aim to partner with existing utility companies [59], who own the grid, to provide a platform that allow users to trade electricity in real-time over the NAD Exchange [58]. As of March 2018 [59], the company has developed a prototype and is currently working on partnering with a utility company to provide a pilot project.

As explained in the NAD Grid whitepaper [48], the platform can be divided into three core components:

- Consortium Blockchain for electricity exchange

- Selective Sourcing
- The Eden token

Each of these components will be further discussed below:

The **Electricity Exchange** is where the electricity is traded. Through the app or website, the real-time market price can be seen, and smart contracts can be initiated between peers. The smart contract automatically executes payments on the blockchain with zero fees. The consortium blockchain keeps track of the balance for each user. All transactions between peers are stored on the blockchain, and the balance for each participant in the transaction will be automatically updated. Users can at any time verify their balance by auditing the ledger.

Selective Sourcing, which is a NAD Grid invention, requires electricity producers to label their energy production with a carbon footprint, based on the amount of greenhouse gases emitted in kilograms per kilowatt-hour. Consumers can then specify the cleanliness of the electricity they wish to purchase. After specifying the carbon footprint grade a consumer wants, the selective sourcing technology will automatically find the cheapest available electricity, based on the constraints.

Eden tokens, which are part of the Ethereum blockchain, are used to pay service fees for the Selective Sourcing service. The consumed electricity is paid in fiat currency. Each token allows a certain amount of electricity to be sourced with Selective Sourcing.

3.3 Power Ledger

The Australian company, Power Ledger, is a peer-to-peer marketplace for renewable energy where neighbors can trade their excess solar power without a middleman [60]. It was founded in 2016, and did its first application beta testing in 2017. Wanting to utilize the smart contract functionality of Ethereum, part of the platform is built on top of the Ethereum blockchain. The following description of the system is based on the Power Ledger whitepaper [61].

The Power Ledger platform consists of two blockchains, each with its own token. The interaction between these two blockchains is illustrated in figure 3.2

The POWR token is traded on the public Ethereum blockchain, this is used by participants to gain access to the platform, and serves as the fuel of the Power Ledger Ecosystem. A user is always required to have a certain amount of POWR tokens in order to be allowed access to the trading Platform. Furthermore, POWR tokens can be traded for Sparkz tokens.

The Sparkz token is used for the actual electricity transactions and exist on a consortium blockchain. The token can be exchanged for local fiat currency on the Application Host, which is also where applications running on the platform can be accessed. In Australia, 1 Sparkz equals 1 cent AUD.

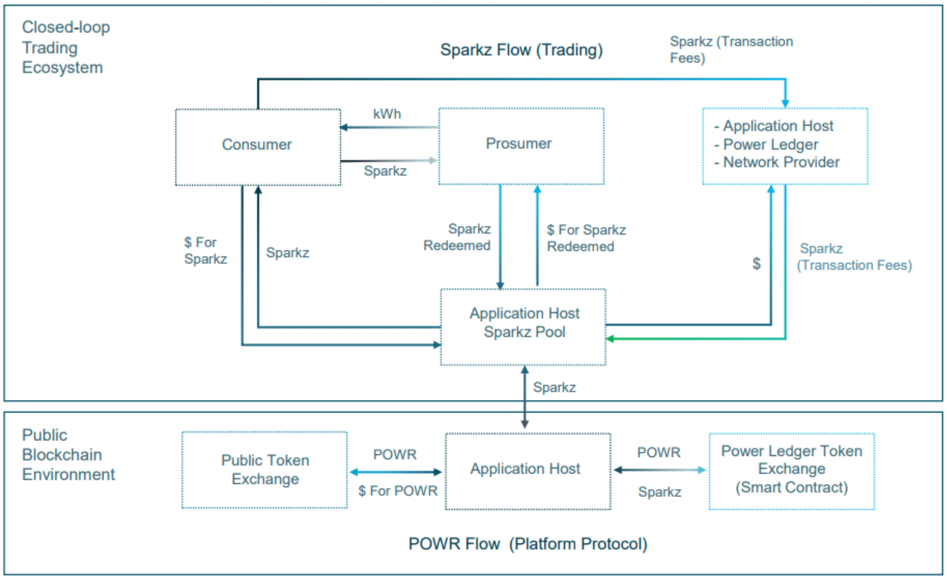


Figure 3.2: Power Ledger Platform [60].

Through the Power Ledger application layer, prosumers and consumers can trade electricity directly with each other. This is done by initiating a smart contract. Meter readings are done by smart meters, and recorded in the blockchain, providing an audit trail for the participants.

Chapter 4

Functional Specifications

This chapter describes the functional specifications of a system for settlement in a microgrid. The specifications are divided into three categories: user interface, application, and the blockchain.

Each node consists of a smart meter for registering consumption and production of electricity. The smart meter is connected to the node's computer. The computer receives data from the smart meter and processes the data before passing it on to the blockchain, which also runs on the computer. Nodes are uniquely identified by their ID.

The system consists of the following participants:

- Consumers purchasing electricity.
- Prosumers who consume electricity, and sell excess electricity from their own production. Prosumers can also purchase electricity from others.
- Public producers delivering electricity on demand through generators, batteries, or power transmitting cables.

Having power delivered by public producers can be a costly solution on e.g. an island, and should be avoided. Participants can be discouraged from this solution by setting extreme prices on electricity from these sources.

4.1 User Interface

This section describes how the user interacts with the system.

1. Users connect to existing nodes on the Network, given an IP address and a port number. The user also chooses a port to run the application on.

2. A website lets users set up smart contracts and tracks their energy flow.
3. The website requires a login where users are identified by their node ID.
4. Prosumers with surplus electricity to sell, put up their availability on the website.
5. Consumers who want to purchase electricity can query available producers and initiate a smart contract.
6. Users can monitor how much electricity they produce and consume in real-time.

4.1.1 User Interface Use Cases

Use cases for how prosumers and consumers interact with the system are described below.

Prosumer

A user who produces excess electricity can choose to sell this electricity to neighbors connected to the microgrid. The user connects to the network grid and is given a unique public/private key pair. The user can log in to the website and advertise the available electricity to other users on the network. When queried by consumers about availability, the users can set up a smart contract and sign it using their private key to create a digital signature. The user can log in to the website and monitor how much electricity they consume, and how much they produce to the grid, as well as the electricity price. If a prosumer does not meet the requirements of a contract, he or she must pay a fee corresponding to the difference between the price the prosumer is selling at, and the price of available energy.

Consumer

Users wishing to purchase electricity from their neighbors can connect to the microgrid network. After logging in to the website with their given node ID and password, they can query other users for available electricity. The consumer can set up a smart contract with a prosumer. Consumers can at any time log in to the website and monitor how much electricity they consume at minute intervals, and at what price. If the consumer decides to use less energy than what was specified in the contract, he or she must pay a fee corresponding to what the producer is losing in profit.

4.2 Application

This section describes how the computer handles incoming data from the users and connected smart meters, forwarding the data to the blockchain, and settling the transaction between users.

1. The computer is connected to a smart meter and receives information about a node's consumption and production of electricity at periodic intervals.
2. The machine passes the readings on to the blockchain for storage.
3. When two users initiate a smart contract, the machine passes it on to the blockchain for validation and storage.
4. New readings from the smart meter are processed on the website and added to the graphs for consumption and production.
5. Settlement is done every time new readings occur. Based on valid contracts in the system and data from the readings, each node's bill is updated reflecting the latest transactions.

4.2.1 Application Use Cases

Following are use cases for when the computer receives a new meter reading, how consumption and production of electricity are monitored, and how smart contracts are processed.

New Meter Reading

Once, every minute, the computer receives measurement readings from the smart meter. The readings include the node ID of the smart meter, and amount of electricity produced and consumed since the last reading. The information is then passed on to the node's blockchain for storage. The settlement module in the applications settles the new readings based on the valid contracts in the system.

Monitoring Consumption and Production

When new measurements are received from the smart meter, the readings become available to the user through the monitoring display on the user interface. A super user can view graphs for all nodes in the network. Individual nodes can only see their own graph, by logging in with the node ID.

New Contract

When two users initiate a contract through the user interface on the web site, they create a digital signature by signing it with their private key. It is then passed to the blockchain for validation and storage. When new meter readings are registered, the system iterates through valid contracts to settle the new transactions

4.3 Blockchain

The blockchain acts as the back-end service to support the requirements given in sections 4.2 and 4.1.

1. The application sends two types of input to the blockchains, a smart contract between two users and electricity transactions. There are two types of blockchains. One for storing smart contracts, and one for storing electricity transactions.
2. The blockchains are distributed between all the nodes in the network.
3. The nodes broadcast their individual transactions across the network, while new smart contracts are sent directly to the consensus leader.
4. The leader of the blockchain proposes a new block when data is available.
5. The proposed block is broadcasted to the consensus nodes, who validate the block and the transactions.
6. If a majority of the nodes deem the block valid, all nodes will add the new block to their blockchain.
7. When the application is settling new transactions, valid smart contracts in the blockchain are triggered and used in the settlement process.
8. The system should scale to work for 50 nodes.

4.3.1 Blockchain Use Cases

How the blockchain handles smart contracts and electricity transactions are described in the use cases below.

Smart Contract

When two users decide to initiate a smart contract, it is sent to the blockchain for validation. The users sign the contract with a digital signature - their private key. The signature can later be verified with the user's public key. When the smart contract is passed to the blockchain module, it is added to a block and verified by the nodes before the block is stored in the blockchain.

Electricity Transactions

New electricity transactions from all network nodes are passed to the blockchain layer every minute. New transactions from each node are broadcasted throughout the network. The current leader adds all new transactions to a block, which is validated by the nodes before it is stored in the blockchain. After being verified, the settlement module settles the transactions.

Chapter 5

Design

The system design is specified in this chapter. The first section illustrates the system architecture, and how the different layers in the system interact on a top-level. The last section is a detailed description of how the main modules in the system should behave.

5.1 System Architecture

The system is divided into two layers, the blockchain layer and the application layer. Each layer consists of multiple modules.

5.1.1 Blockchain Layer

The relation between the modules of the blockchain layer can be seen in figure 5.1. The main modules of the system are the *node*, *consensus* and, *network* modules. Whereas the *storage*, *messages*, and *block* modules are helper modules. The *node* object in the *node* module is a subclass of the *PeerManager* in the *network* module, which allows the *node* object to directly use methods in the *network* module to communicate with other nodes.

The *node* object controls the main process in the system, including the consensus process, which is initiated by the *node* object. Proposing new blocks and adding validated blocks to the blockchain is also controlled by the *node* object. The *network* module handles the process of sending and receiving network messages, before they are passed to the *node* object for further processing. The *message* module acts as a helper class, which contains all the different types of messages that are used in the system. The *consensus* module handles everything related to the consensus process,

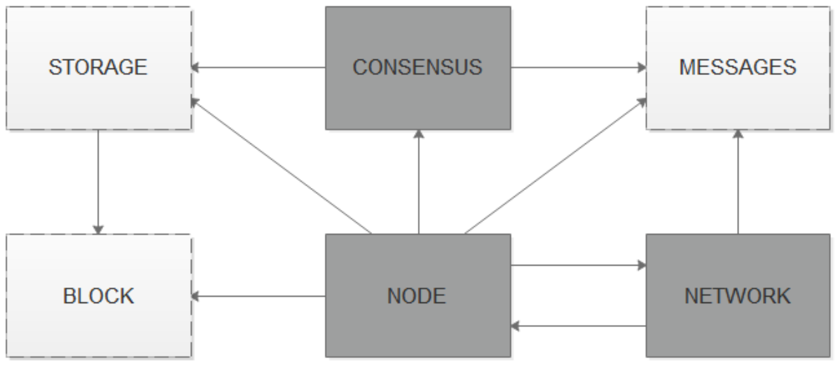


Figure 5.1: Connection between modules in blockchain layer.

including validation of blocks. The *storage* helper module is where the blockchain, and all other information that needs to be written to disk, is stored. New *block* objects from the *block* module are created by the *node* object. The *block* helper module is also used by the *storage* module for converting blocks written to file as strings, back to block objects.

5.1.2 Application Interaction with Blockchain

The modules in the blockchain layer are abstracted into a single blockchain module in figure 5.2. The *electricity data* is read from the smart meters, and passed into the system. The data is processed on the *user interface*, so it can be monitored by the users. The data is also passed to the *blockchain* module for storage. *Smart contracts* are created by users in the *user interface*, before they are passed to the *blockchain* for storage. Based on the smart contracts and electricity data that is stored in the blockchain, the *settlement* process is executed.

5.2 Module Design

The individual modules are explained in further detail in this section.

5.2.1 Blockchain

There are two blockchains for the microgrid, one for storing all electricity transactions in the system, and one for storing all smart contracts. The blockchains are the back-end services that enable the settlement process.

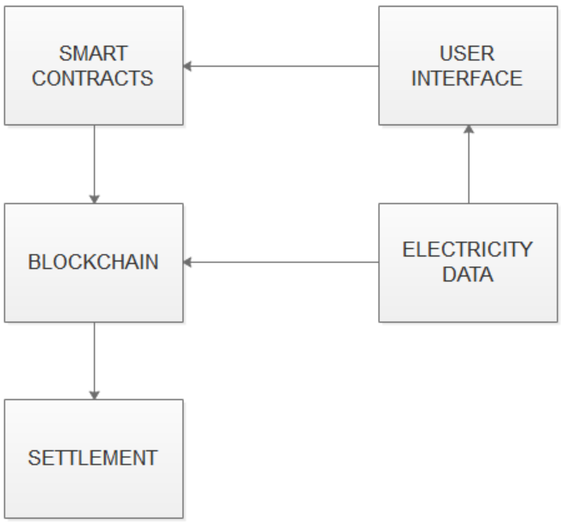


Figure 5.2: Interaction between application and blockchain.

Consensus

The consensus protocol is modeled after the Raft protocol. All nodes are part of the consensus process, and any node can become leader. Nodes in the system act as the clients and send their transactions and contracts to the consensus leader, who then initiates the validation process by creating a block. If consensus is reached among the validating nodes, the block is stored in one of the blockchains, depending on if it contains electricity transaction or a smart contract.

In Raft, the log contain operations that should be executed on a state machine. Consensus is needed in order for all nodes to execute the same operations in the same order. Some modifications has been done to this protocol in order for it to be suitable in a blockchain environment. Instead of a operations log, a *proposed blocks* log is used. The log consists of all blocks the leader has proposed, awaiting to be validated and possibly commited to the blockchain. The blockchain log is the equivalent to the state machine where all nodes must have the same block entries in the same order.

5.2.2 Smart Contract

Smart contracts enable automatic transfer of assets, once conditions are met. Thus, in contrast to traditional contracts, smart contracts do not only define the terms and conditions of an agreement, they also provide a method for enforcing the agreement.

When a consumer and prosumer initiate a smart contract, the terms of the contract must include the following.

1. The start time of the contract.
2. The end time of the contract.
3. The ID's of the seller and buyer.
4. The minimum amount of electricity transacted between the parties of the contract.
5. Digital signatures from both parties of the contract.

The smart contract API is callable from the website, and is triggered when two parties initiate the contract. Any node in the system can only have one active contract at a time. New transactions trigger settlement based on smart contract.

5.2.3 Settlement

Energy trading is a very complex task; thus, four assumptions are made for the settlement module in this system:

- During any interval, the amount of electricity consumed in the system equals the amount of electricity produced in the system.
- Any participant in the network has one, and only one, contract with another participant in the network at any given time.
- If the consumer of a contract has consumed more electricity than the producer of the contract has produced, the excess consumed electricity is produced by the pure producers in the system.
- If a prosumer produces more electricity than is consumed by itself or the consumer in the contract, during an interval, it is assumed that the electricity is stored in batteries and not part of the settlement process.

If prosumer A has a smart contract with consumer B, and the system registers that these nodes have produced and consumed electricity during any interval in the duration of the contract, node B's bill will register that it owes node A for the consumed electricity of the period, and likewise node A's bill will show that node B owes money for the electricity. The bills are accumulative and assumed settled at regular intervals, e.g. each month.

5.2.4 User Interface

The user interface is a website. A user is either a prosumer or a consumer in the microgrid, or the special case of a super user, who can access all the information of the system. Users may log in to their own profile to monitor real-time, and past, consumption and production of electricity. All users are given a public/private key pair when they join the network. Furthermore, the website has a trading platform where prosumers advertise how much electricity they are selling. Consumers can inquire the prosumers to make a contract. The contract is finalized and sent to the consensus leader when both parties have signed the contract with their private keys.

Chapter 6

Implementation

This chapter describes the implementation of the system in this thesis. The first section details the software used in the implementation. The rest of the chapter describes how the individual modules of the system are implemented.

6.1 Software

Following is a list of the resources used in this system implementation:

- Python for implementation of the system
- Python framework Twisted for network communication
- Flask for web applications
- ECDSA for digital signatures and verification

6.1.1 Python

Python [62] was chosen as the programming language for the system implementation in this thesis. This is due to Pythons multipurpose abilities, and the intuitive syntax and programming style, which enables a rapid program development. Python also possesses characteristics like object-orientation, as well as being modular and dynamic. There exists a wide range of modules and libraries in Python.

6.1.2 Twisted

Twisted [63] is a networking engine written in Python. Some of the main components of the Twisted library are described below:

- **Reactor**: The reactor reacts to events in a loop and dispatches them to predetermined callback functions that handle the events. The event loop runs endlessly, unless it is told to stop.
- **Protocols**: Each protocol object in Twisted represents one connection. Protocols handle network events in an asynchronously manner. The protocol is responsible for handling incoming data, and connections to other peers in the network.
- **Factory**: Protocol instances are created in the factory, one for each connection. The factory utilizes the protocol for communication with its peers. Information that is persistent across connections is stored in the factory.
- **Transport**: A transport is a method that represents the actual connection between the two endpoints in a protocol, e.g. a TCP connection. The transport is used for communication between the two endpoints, as it writes data from one connection to the other.

6.1.3 Flask

Flask is a microframework for web development written in Python [64]. Flask is a lightweight WSGI (Web Server Gateway Interface) web application framework that is designed for quick development. The Flask API is easy to use due to the frameworks extensive documentation, and since there are no additional tools or libraries required to run the web application.

6.1.4 ECDSA

The ECDSA (Elliptic Curve Digital Signature Algorithm) library is an implementation of the ECDSA cryptography algorithm written in Python [65]. The library creates key pairs for signing messages and verifying the signatures. Signatures are created by using the private signing key on the message that is to be signed. The signature is verified by using the public verifying key, which is derived from the signing key. The verification method ensures that the unsigned message is the same as the message signed with the signing key.

6.2 Blockchain Layer

This section describes the main modules implemented in the blockchain layer.

6.2.1 Node Module

The node module consists of a *Node* class, which is a subclass of the *PeerManager* class. The main component of the *Node* class is the state machine, which is triggered by the reactor, which makes a *LoopingCall* at periodic intervals.

Every time the *state_machine* method is executed, the network leader sends out an *append entries* RPC to all its followers. This lets the followers know that the leader is still operating. If new transactions are available, the leader creates a new block including these transactions, and proposes the block to its followers. If a follower finds a block valid, its stores the block in its log of proposed blocks during the next execution of the *state machine* method.

The leader keeps a timer on the proposed block. If a timeout occurs before a majority has validated and accepted the block, the leader steps down, and a new election for a leader will start once the *leader election timeout* occurs. However, if the majority validates and accepts the block before the timeout occurs, the leader will add the block to the blockchain, and notify its followers to do the same.

6.2.2 Network Module

A P2P network is implemented in the network module, using the Twisted [63] framework.

The initial network node starts a server, and a client that connects to the server. Other nodes also start a server on a specified port number and a client which connects to a server already running on a given IP address and port number.

The module consists of two classes, a *PeerManager* and a *Peer*. The *PeerManager* class is a Twisted factory, and is responsible for storing information about the peer, which is persistent between connections. This includes attributes such as a dictionary containing all connections to other peers, methods for adding and removing peers to the dictionary, as well as a method for starting a new client that connects to a new peer's server.

Peer is a subclass of the Twisted protocol *IntNStringReceiver*. This means that each received message is a callback to the method *stringReceived*. The *Peer* class also keeps track of information in a connection between two peers. This includes a method for discovering when the connection is lost. The main method of the *Peer* class is the *stringReceived* method. Based on what message type that is received, the method decides what to do with the message.

The initial message sent by a client, node A, connecting to a new server, node B, is the *hello* message. This includes the client's node ID, IP address, and host port which is information the server on node B keeps track of for all its peers. If the connection is successful, the server on node B acknowledges the message by sending its own node ID, IP address, and port number for node A to store. Server B proceeds by sending a message containing information about all its peers to node

A. Node A then starts a new client for all the peers it is not already connected to and repeats the process described above.

Other messages are processed in the factory, and further handled by the *Node* object.

6.2.3 Consensus Module

The consensus module used in this system is Raft, and consists of a *Validator* class. A *Validator* object is created by the *Node* object, to handle the validation and consensus in the blockchain. Once a *Validator* object is created, a *leader election timer* starts. The timeout is cancelled if the node receives a message from a leader. If the timeout occurs, the *start leader election* method is called. A node promotes itself to a candidate, starts a new term and votes for itself as leader before requesting votes from its peers with the *request vote* RPC.

A follower receiving a *request vote* RPC will vote for the candidate if it has not already voted in that term or voted for someone else. The follower will also verify that the candidate's blockchain log is at least as up to date as its own before casting the vote. If the candidate receives a majority vote it will establish itself as leader by sending out an *append entries* RPC.

Another possible outcome of the election process is that the candidate does not receive a majority vote before the election timeout occurs. It will then either promote itself as a candidate and start a new term, or receive a *request vote* RPC from another candidate, who has started a new term.

The third outcome is that several nodes become candidate at the same time. If a candidate receives an *append entries* RPC from a different node, it will step down and become a follower.

As previously mentioned, leaders send out *append entries* RPCs from the *Node* object. Followers respond to the RPC in the *Validator* object. With every RPC, the leader includes the previous log index and log term. In the *respond append entries* method, a follower checks if it has this entry in its log and responds accordingly. If there is no entry, the leader decreases the index until a match is found. If there is a conflicting entry, the follower deletes its own entry and writes the leaders entry in the log. This process ensures that all nodes have the same blockchain, and that all nodes are up to date with the blockchain.

6.2.4 Block Module

The block module consists of a *Block* class. The attributes of the class are: the index of the block, which is the number of the block in the blockchain where the genesis block starts at 0; the hash of the block immediately preceding the block; a timestamp of when the block was created; the transactions included in the block; and the new hash of the block.

The block hash is calculated using the python hashlib [66] library. It uses SHA-256 hash algorithm based on the index, previous hash, timestamp and transactions in the block, thus creating a unique hash for every block.

6.2.5 Storage Module

The storage module reads from and writes to file everything that needs to be persistent in case of a system shutdown. This information includes a nodes node ID, public/private keys, and vote information. Furthermore, the blockchains are stored to file by the storage module, as well as the *proposed block* log.

6.3 Application Layer

This section includes descriptions of modules implemented in the application layer of the system.

6.3.1 Settlement

Settlement occurs based on existing smart contracts in the system. The protocol is triggered by the blockchain leader every time there is new transaction information in the system that has been verified and added to a block in the blockchain. The settlement algorithm that is implemented is fairly naive, and based around four assumptions given in section 5.2.3. An outline of the algorithm can be seen in Algorithm 1.

6.3.2 Smart Contracts

The smart contract API was not developed. Contracts were created manually as a PoC in the user interface, and the settlement module.

However, the process of signing and verifying the contracts was tested. The contract was signed with the private key, obtained by the ECDSA library, of each node party of the contract. Verification was done with the verification method of the public key.

6.3.3 Electricity Data

As the system is not connected to any hardware, it is only tested using simulated data. Each node object reads the electricity data at periodic intervals.

Algorithm 1: Naive settlement algorithm

```
Result: wallet accounts for all nodes updated
buyer bought
foreach contract do
    if seller consumed more electricity than produced then
        seller bought required electricity from pure producer
        buyer bought required electricity from pure producer
    else if seller sold is 0 and buyer bought is more than 0 then
        buyer bought all electricity from pure producer
    else
        if seller sold more than buyer bought then
            buyer bought all required electricity from seller
            assume excess energy stored on battery
        else
            buyer bought all electricity seller sold
            if buyer required more electricity then
                buyer bought required electricity from pure producer
end
```

6.3.4 User Interface

The user interface was not integrated with the rest of the system, rather a interface layout was outlined as seen from the consumers perspective. The interface consisted of a web page with two menu options: one for monitoring consumed and produced electricity in real-time; and one to create smart contracts based on prosumer availability.

Chapter 7

Testing

This chapter details how the system was tested. Section 7.1 describes the test case for the complete system, while section 7.2 specifies how some of the individual modules of the system were tested.

7.1 System Test

A functional test was run on the complete system, using pseudo-random simulation data. The test approach is summarized below:

- Network consisted of four prosumer nodes with 48-hours worth of test data for produced and consumed electricity, given in kWh per minute. Each node periodically read the test data from a CSV (Comma Separated Value) file. The four nodes were run as different processes on the same computer.

- Smart contracts were manually constructed in pairs between the node, with one node selling, and one node buying electricity per contract.

- Transactions were settled according to the settlement algorithm described in section 6.3.1. For simplicity, the price of 1 kWh was set to 1 NOK, regardless of the production source.

- All transactions were stored in the blockchain. Transactions from all nodes during a given interval were stored in the same block.

7.2 Module Testing

7.2.1 User Interface

Tests were executed on the implemented functionality in the user interface. The monitoring of consumed and produced electricity of a node was tested using simulated data stored in a CSV file.

For the smart contract segment of the user interface, a simple table of producer availability was constructed using arbitrary data, and a form for consumers to initiate a contract was also created.

7.2.2 Network Module

The ability for nodes to communicate over the network is crucial for the system to function. Several aspects of this module were tested:

- Communication between two machines on the same network.
- Communication between two machines on two separate networks.
- Processing of different message types.
- Ability to send message to the correct recipient.

7.2.3 Consensus

The consensus module is an important feature in order for the system to function correctly. The following tests were performed to verify this:

- During normal operation, the consensus module was tested on 10 nodes, running as separate processes on two separate computers.
- Adding new nodes under operation.
- Election of new leader after a leader has stepped down.
- Ability to correct wrong data on a node.

7.2.4 Smart Contract

Since the smart contract API was not fully implemented, only the digital signatures were tested. The test consisted of two nodes, a public and private key was generated for both nodes using the ECDSA library. Both nodes signed an arbitrary message using their private keys. Both signatures were then verified using the node's public keys. An additional test was conducted where the signature was signed with one nodes private key, and attempted verified with the other nodes public key. The source code for this test can be found in appendix A

Chapter 8

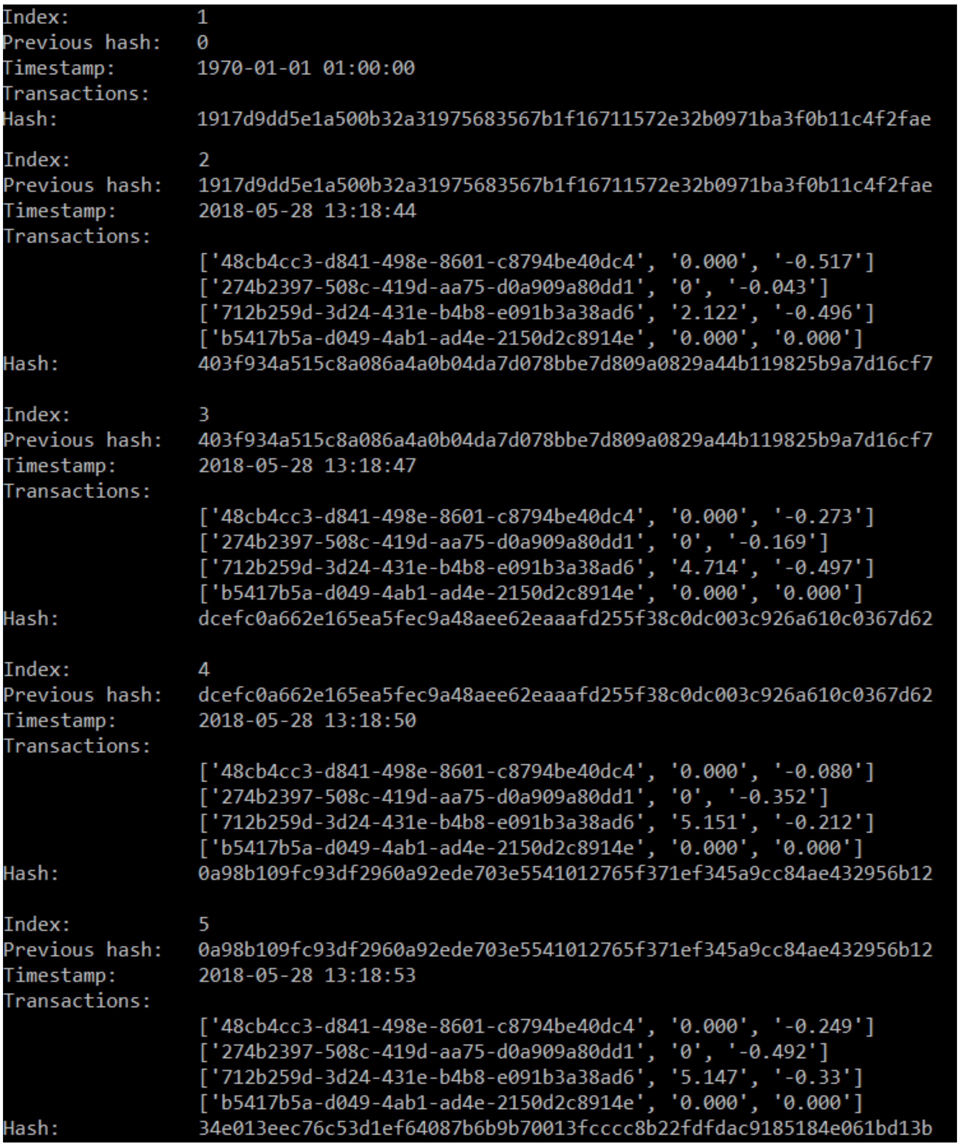
Result

The results from the tests described in chapter 7 are presented in this chapter. The results are further discussed in chapter 9.

8.1 System Results

After running the full system on 48-hours worth of test data, each node produced a blockchain with 2881 blocks, where each block contained four transactions - one for each node. This resulted in a 1.22 MB sized file, which roughly equals 423 bytes per block. The complete result of the test can be found with the source code. To ensure that correct behavior of the system, an extract of the data is shown in figures 8.1 and 8.2. This result is based on the transaction data shown in appendix B. The blockchain in figure 8.1 shows the five first blocks in the chain, starting with the genesis block. Each block contains the index of the block, the time at which it was created, the hash of the previous block in the chain, the transaction data for each node from the previous period, and the hash of the current block. The transaction data contains the node ID, consumed electricity and produced electricity in kWhs, respectively, for each node. Compared to the transactions in appendix B, each block contains the correct transactions. However, the timestamp of the block does not match the time of the transaction data, as this is only simulated data.

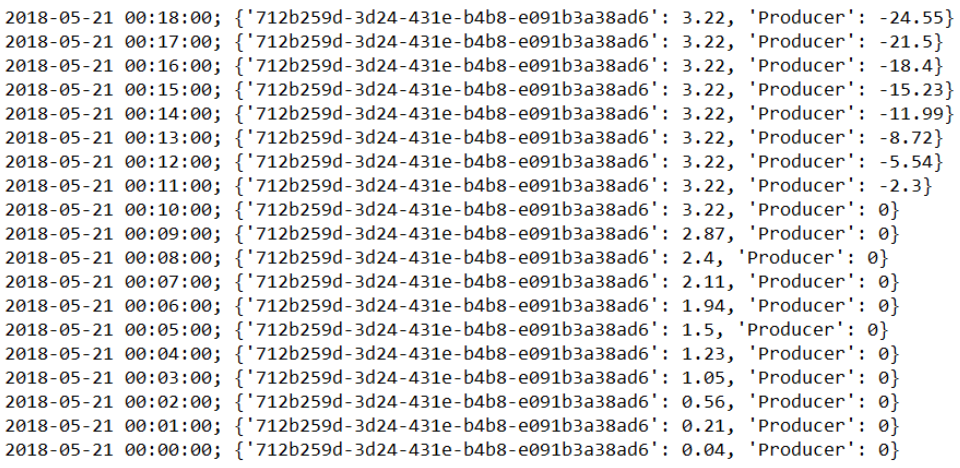
Figure 8.2 shows the settlement between two of the nodes in the network. Each line represents the accumulated settlement over time, with the most recent settlement at the top. Negative numbers represent money that is owned to the specified node, by the node whose settlement account is shown. Likewise, positive numbers represent money that the specified node owes the node whose settlement account is shown.



```
Index:          1
Previous hash:  0
Timestamp:      1970-01-01 01:00:00
Transactions:
Hash:           1917d9dd5e1a500b32a31975683567b1f16711572e32b0971ba3f0b11c4f2fae

Index:          2
Previous hash:  1917d9dd5e1a500b32a31975683567b1f16711572e32b0971ba3f0b11c4f2fae
Timestamp:      2018-05-28 13:18:44
Transactions:
                ['48cb4cc3-d841-498e-8601-c8794be40dc4', '0.000', '-0.517']
                ['274b2397-508c-419d-aa75-d0a909a80dd1', '0', '-0.043']
                ['712b259d-3d24-431e-b4b8-e091b3a38ad6', '2.122', '-0.496']
                ['b5417b5a-d049-4ab1-ad4e-2150d2c8914e', '0.000', '0.000']
Hash:           403f934a515c8a086a4a0b04da7d078bbe7d809a0829a44b119825b9a7d16cf7

Index:          3
Previous hash:  403f934a515c8a086a4a0b04da7d078bbe7d809a0829a44b119825b9a7d16cf7
Timestamp:      2018-05-28 13:18:47
Transactions:
                ['48cb4cc3-d841-498e-8601-c8794be40dc4', '0.000', '-0.273']
                ['274b2397-508c-419d-aa75-d0a909a80dd1', '0', '-0.169']
                ['712b259d-3d24-431e-b4b8-e091b3a38ad6', '4.714', '-0.497']
                ['b5417b5a-d049-4ab1-ad4e-2150d2c8914e', '0.000', '0.000']
Hash:           dcefc0a662e165ea5fec9a48aee62eaaafd255f38c0dc003c926a610c0367d62

Index:          4
Previous hash:  dcefc0a662e165ea5fec9a48aee62eaaafd255f38c0dc003c926a610c0367d62
Timestamp:      2018-05-28 13:18:50
Transactions:
                ['48cb4cc3-d841-498e-8601-c8794be40dc4', '0.000', '-0.080']
                ['274b2397-508c-419d-aa75-d0a909a80dd1', '0', '-0.352']
                ['712b259d-3d24-431e-b4b8-e091b3a38ad6', '5.151', '-0.212']
                ['b5417b5a-d049-4ab1-ad4e-2150d2c8914e', '0.000', '0.000']
Hash:           0a98b109fc93df2960a92ede703e5541012765f371ef345a9cc84ae432956b12

Index:          5
Previous hash:  0a98b109fc93df2960a92ede703e5541012765f371ef345a9cc84ae432956b12
Timestamp:      2018-05-28 13:18:53
Transactions:
                ['48cb4cc3-d841-498e-8601-c8794be40dc4', '0.000', '-0.249']
                ['274b2397-508c-419d-aa75-d0a909a80dd1', '0', '-0.492']
                ['712b259d-3d24-431e-b4b8-e091b3a38ad6', '5.147', '-0.33']
                ['b5417b5a-d049-4ab1-ad4e-2150d2c8914e', '0.000', '0.000']
Hash:           34e013eec76c53d1ef64087b6b9b70013fcccc8b22fdfdac9185184e061bd13b
```

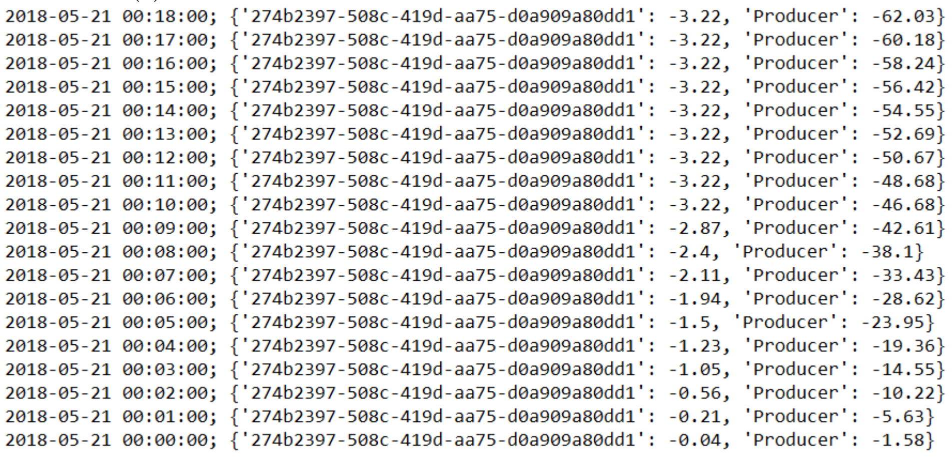
Figure 8.1: Five first blocks in the blockchain.

The results in figure 8.2 show that node 712b259d-3d24-431e-b4b8-e091b3a38ad6, the buyer, owes node 274b2397-508c-419d-aa75-d0a909a80dd1, the seller, 3.22 NOK after 19 minutes. Both nodes have consumed a considerable amount of electricity generated from the pure producers. The result of the settlement in figure 8.2 is manually verified in figure 8.3. The verification shows the same result as the system test, thus verifying that the settlement algorithm produced the correct result.



```
2018-05-21 00:18:00; {'712b259d-3d24-431e-b4b8-e091b3a38ad6': 3.22, 'Producer': -24.55}
2018-05-21 00:17:00; {'712b259d-3d24-431e-b4b8-e091b3a38ad6': 3.22, 'Producer': -21.5}
2018-05-21 00:16:00; {'712b259d-3d24-431e-b4b8-e091b3a38ad6': 3.22, 'Producer': -18.4}
2018-05-21 00:15:00; {'712b259d-3d24-431e-b4b8-e091b3a38ad6': 3.22, 'Producer': -15.23}
2018-05-21 00:14:00; {'712b259d-3d24-431e-b4b8-e091b3a38ad6': 3.22, 'Producer': -11.99}
2018-05-21 00:13:00; {'712b259d-3d24-431e-b4b8-e091b3a38ad6': 3.22, 'Producer': -8.72}
2018-05-21 00:12:00; {'712b259d-3d24-431e-b4b8-e091b3a38ad6': 3.22, 'Producer': -5.54}
2018-05-21 00:11:00; {'712b259d-3d24-431e-b4b8-e091b3a38ad6': 3.22, 'Producer': -2.3}
2018-05-21 00:10:00; {'712b259d-3d24-431e-b4b8-e091b3a38ad6': 3.22, 'Producer': 0}
2018-05-21 00:09:00; {'712b259d-3d24-431e-b4b8-e091b3a38ad6': 2.87, 'Producer': 0}
2018-05-21 00:08:00; {'712b259d-3d24-431e-b4b8-e091b3a38ad6': 2.4, 'Producer': 0}
2018-05-21 00:07:00; {'712b259d-3d24-431e-b4b8-e091b3a38ad6': 2.11, 'Producer': 0}
2018-05-21 00:06:00; {'712b259d-3d24-431e-b4b8-e091b3a38ad6': 1.94, 'Producer': 0}
2018-05-21 00:05:00; {'712b259d-3d24-431e-b4b8-e091b3a38ad6': 1.5, 'Producer': 0}
2018-05-21 00:04:00; {'712b259d-3d24-431e-b4b8-e091b3a38ad6': 1.23, 'Producer': 0}
2018-05-21 00:03:00; {'712b259d-3d24-431e-b4b8-e091b3a38ad6': 1.05, 'Producer': 0}
2018-05-21 00:02:00; {'712b259d-3d24-431e-b4b8-e091b3a38ad6': 0.56, 'Producer': 0}
2018-05-21 00:01:00; {'712b259d-3d24-431e-b4b8-e091b3a38ad6': 0.21, 'Producer': 0}
2018-05-21 00:00:00; {'712b259d-3d24-431e-b4b8-e091b3a38ad6': 0.04, 'Producer': 0}
```

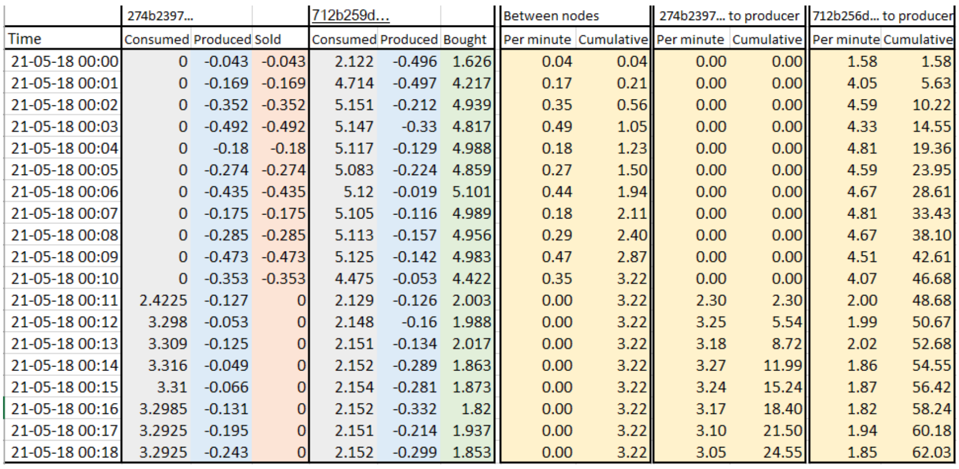
(a) Settlement for node 274b2397-508c-419d-aa75-d0a909a80dd1



```
2018-05-21 00:18:00; {'274b2397-508c-419d-aa75-d0a909a80dd1': -3.22, 'Producer': -62.03}
2018-05-21 00:17:00; {'274b2397-508c-419d-aa75-d0a909a80dd1': -3.22, 'Producer': -60.18}
2018-05-21 00:16:00; {'274b2397-508c-419d-aa75-d0a909a80dd1': -3.22, 'Producer': -58.24}
2018-05-21 00:15:00; {'274b2397-508c-419d-aa75-d0a909a80dd1': -3.22, 'Producer': -56.42}
2018-05-21 00:14:00; {'274b2397-508c-419d-aa75-d0a909a80dd1': -3.22, 'Producer': -54.55}
2018-05-21 00:13:00; {'274b2397-508c-419d-aa75-d0a909a80dd1': -3.22, 'Producer': -52.69}
2018-05-21 00:12:00; {'274b2397-508c-419d-aa75-d0a909a80dd1': -3.22, 'Producer': -50.67}
2018-05-21 00:11:00; {'274b2397-508c-419d-aa75-d0a909a80dd1': -3.22, 'Producer': -48.68}
2018-05-21 00:10:00; {'274b2397-508c-419d-aa75-d0a909a80dd1': -3.22, 'Producer': -46.68}
2018-05-21 00:09:00; {'274b2397-508c-419d-aa75-d0a909a80dd1': -2.87, 'Producer': -42.61}
2018-05-21 00:08:00; {'274b2397-508c-419d-aa75-d0a909a80dd1': -2.4, 'Producer': -38.1}
2018-05-21 00:07:00; {'274b2397-508c-419d-aa75-d0a909a80dd1': -2.11, 'Producer': -33.43}
2018-05-21 00:06:00; {'274b2397-508c-419d-aa75-d0a909a80dd1': -1.94, 'Producer': -28.62}
2018-05-21 00:05:00; {'274b2397-508c-419d-aa75-d0a909a80dd1': -1.5, 'Producer': -23.95}
2018-05-21 00:04:00; {'274b2397-508c-419d-aa75-d0a909a80dd1': -1.23, 'Producer': -19.36}
2018-05-21 00:03:00; {'274b2397-508c-419d-aa75-d0a909a80dd1': -1.05, 'Producer': -14.55}
2018-05-21 00:02:00; {'274b2397-508c-419d-aa75-d0a909a80dd1': -0.56, 'Producer': -10.22}
2018-05-21 00:01:00; {'274b2397-508c-419d-aa75-d0a909a80dd1': -0.21, 'Producer': -5.63}
2018-05-21 00:00:00; {'274b2397-508c-419d-aa75-d0a909a80dd1': -0.04, 'Producer': -1.58}
```

(b) Settlement for node 712b259d-3d24-431e-b4b8-e091b3a38ad6

Figure 8.2: Settlement between two prosumer nodes that have initiated a smart contract.



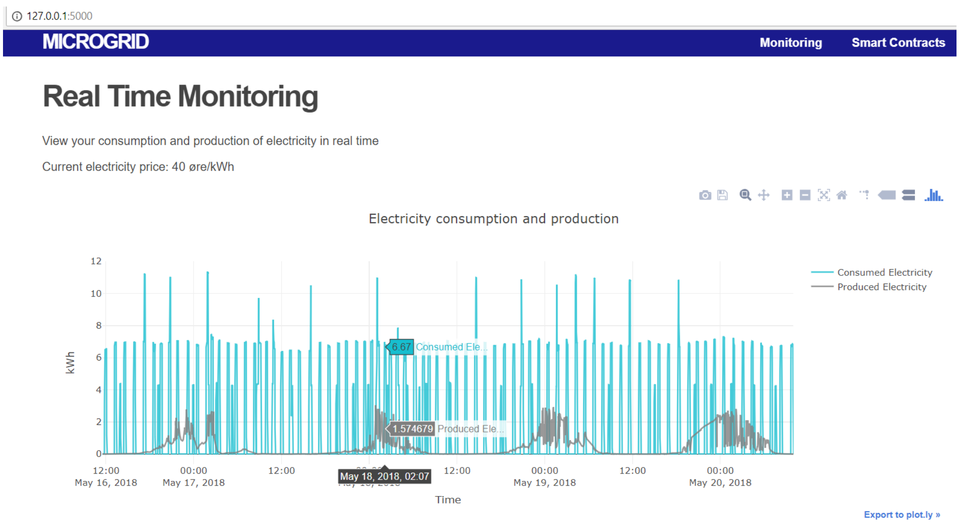
	274b2397...			712b259d...			Between nodes		274b2397... to producer		712b256d... to producer	
Time	Consumed	Produced	Sold	Consumed	Produced	Bought	Per minute	Cumulative	Per minute	Cumulative	Per minute	Cumulative
21-05-18 00:00	0	-0.043	-0.043	2.122	-0.496	1.626	0.04	0.04	0.00	0.00	1.58	1.58
21-05-18 00:01	0	-0.169	-0.169	4.714	-0.497	4.217	0.17	0.21	0.00	0.00	4.05	5.63
21-05-18 00:02	0	-0.352	-0.352	5.151	-0.212	4.939	0.35	0.56	0.00	0.00	4.59	10.22
21-05-18 00:03	0	-0.492	-0.492	5.147	-0.33	4.817	0.49	1.05	0.00	0.00	4.33	14.55
21-05-18 00:04	0	-0.18	-0.18	5.117	-0.129	4.988	0.18	1.23	0.00	0.00	4.81	19.36
21-05-18 00:05	0	-0.274	-0.274	5.083	-0.224	4.859	0.27	1.50	0.00	0.00	4.59	23.95
21-05-18 00:06	0	-0.435	-0.435	5.12	-0.019	5.101	0.44	1.94	0.00	0.00	4.67	28.61
21-05-18 00:07	0	-0.175	-0.175	5.105	-0.116	4.989	0.18	2.11	0.00	0.00	4.81	33.43
21-05-18 00:08	0	-0.285	-0.285	5.113	-0.157	4.956	0.29	2.40	0.00	0.00	4.67	38.10
21-05-18 00:09	0	-0.473	-0.473	5.125	-0.142	4.983	0.47	2.87	0.00	0.00	4.51	42.61
21-05-18 00:10	0	-0.353	-0.353	4.475	-0.053	4.422	0.35	3.22	0.00	0.00	4.07	46.68
21-05-18 00:11	2.4225	-0.127	0	2.129	-0.126	2.003	0.00	3.22	2.30	2.30	2.00	48.68
21-05-18 00:12	3.298	-0.053	0	2.148	-0.16	1.988	0.00	3.22	3.25	5.54	1.99	50.67
21-05-18 00:13	3.309	-0.125	0	2.151	-0.134	2.017	0.00	3.22	3.18	8.72	2.02	52.68
21-05-18 00:14	3.316	-0.049	0	2.152	-0.289	1.863	0.00	3.22	3.27	11.99	1.86	54.55
21-05-18 00:15	3.31	-0.066	0	2.154	-0.281	1.873	0.00	3.22	3.24	15.24	1.87	56.42
21-05-18 00:16	3.2985	-0.131	0	2.152	-0.332	1.82	0.00	3.22	3.17	18.40	1.82	58.24
21-05-18 00:17	3.2925	-0.195	0	2.151	-0.214	1.937	0.00	3.22	3.10	21.50	1.94	60.18
21-05-18 00:18	3.2925	-0.243	0	2.152	-0.299	1.853	0.00	3.22	3.05	24.55	1.85	62.03

Figure 8.3: Manual verification of correctness of settlement protocol.

8.2 Module Results

8.2.1 User Interface

The results from the tests run on the user interface module propose a layout for the user interface. Figure 8.4 shows how the consumption and production of electricity can be viewed in real-time for each node. Figure 8.5 show how the producers can advertise their available excess electricity, and an approach for creating smart contracts from the buyer's point of view.


127.0.0.1:5000

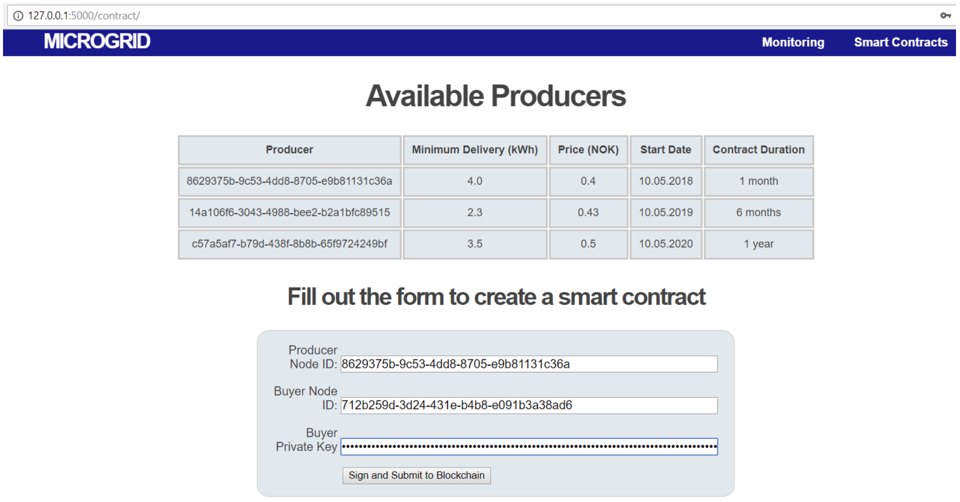
MICROGRID Monitoring Smart Contracts

Real Time Monitoring

View your consumption and production of electricity in real time

Current electricity price: 40 øre/kWh

Figure 8.4: Real-time monitoring of consumed and produced electricity displayed on user interface.


127.0.0.1:5000/contract/

MICROGRID Monitoring Smart Contracts

Available Producers

Producer	Minimum Delivery (kWh)	Price (NOK)	Start Date	Contract Duration
8629375b-9c53-4dd8-8705-e9b81131c36a	4.0	0.4	10.05.2018	1 month
14a106f6-3043-4988-bee2-b2a1bfc89515	2.3	0.43	10.05.2019	6 months
c57a5af7-b79d-438f-8b8b-65f9724249bf	3.5	0.5	10.05.2020	1 year

Fill out the form to create a smart contract

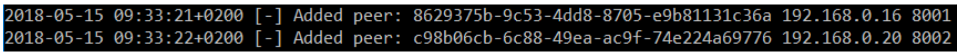
Producer Node ID:	8629375b-9c53-4dd8-8705-e9b81131c36a
Buyer Node ID:	712b259d-3d24-431e-b4b8-e091b3a38ad6
Buyer Private Key	••

Sign and Submit to Blockchain

Figure 8.5: Availability producers selling excess electricity.

8.2.2 Network Module

Figure 8.6 shows a node connected to two other nodes on different IP addresses. This shows that the system is able to communicate on two separate machines on the same network.



```
2018-05-15 09:33:21+0200 [-] Added peer: 8629375b-9c53-4dd8-8705-e9b81131c36a 192.168.0.16 8001
2018-05-15 09:33:22+0200 [-] Added peer: c98b06cb-6c88-49ea-ac9f-74e224a69776 192.168.0.20 8002
```

Figure 8.6: Network communication between two computers on the same network.

However, when the network module was tested on computers connected to separate networks, it failed. Due to issues with the firewall and port forwarding on the routers, these problems were not further evaluated.

All the different messages in the network include the message type. After having established a connection between two nodes on the network, one node tried to send every type of message to the other node in the network which resulted in the correct response from the other node.

Messages in the network are either broadcasted or sent to a specific node. The tests showed that broadcasted messages reached all nodes connected to the network. Furthermore, messages could be sent to any single node where a Twisted connection was established between the nodes.

8.2.3 Consensus

The result from running ten nodes during normal operations provided successful results. The network was able to elect a leader, which in turn proposed new blocks to the network based on transaction data from all of the nodes. The proposed blocks were correctly validated by the network and added to the blockchain of all the nodes.

The process of adding new nodes after the network had started operating under normal conditions, gave ambiguous results. During the tests with ten nodes, an eleventh node was added to the network after the leader was elected and blocks were being added to the blockchain. This resulted in incorrect behavior for the eleventh node, as it was unable to add any of the blocks the rest of the network had agreed on. Rather it was stuck on the genesis block. However, when the same test was done on four nodes and a fifth node being added to the network after some time of operation, the fifth node was able to receive all the blocks added to the network before it joined. Furthermore, the node continued normal operations when it was caught up.

To test if the network was able to elect a new leader after the current network leader had crashed, the process running the leader was terminated. A new leader was elected according to the timer of the leader timeout function. The duration of this timer was adjusted, and resulted in coherent election time. Furthermore, the leader process was terminated and re-connected to the network. This was done multiple times, until each node in the entire network of ten nodes had been elected leader.

An incorrect entry in the *proposed blocks* log of a node was manually entered. This test resulted in two findings. First, this node was unable to be elected leader, due to errors in the log. Second, when the consensus leader found the incorrect entry, it replaced the log entry with the correct entry which was then added to the blockchain.

8.2.4 Smart Contract

The first two tests, where the signatures were verified by the public key corresponding to the private key that was used to sign the messaged, resulted in *True* verification. Furthermore, the final test where a public key not corresponding to the private key was used to verify the signature, raised a *BadSignatureError*, proving the correct result.

Chapter 9

Discussion

9.1 Interpretation of Results

The results from the system test show that a blockchain may be used to settle the value of the energy flowing in the system.

The results from the test of the complete system were positive. The extracted data set that was manually verified proved to be correct. However, it is difficult to prove correctness of all the 2880 data entries. Further testing needs to be conducted on the system. These tests include, but are not limited to, testing of corner cases, and perhaps most important, the system must be connected to actual hardware and tested on real transaction data from energy flow in the system.

Apart from the failure of connection across multiple networks, the network protocol showed good results. This module is a fundamental feature in order for the system to behave correctly. The results from the test of the system as a whole, and the modules built on-top of this protocol indicate that it works well.

The inconsistent results from the consensus tests, indicate that there may be an issue with the scalability of the network. If issues occur already at ten nodes, the proposed consensus algorithm will not be applicable in an actual system with perhaps as many as fifty nodes.

Although a smart contract API was not developed in this thesis, the tests of the digital signatures provide a good starting point for further development of smart contracts

9.2 Evaluation of system

In this section, the system implementation will first be evaluated based on the functional specifications presented in chapter 4, and later compared to existing implementations presented in chapter 3

9.2.1 Evaluation of Functional Specifications

User Interface

Of the six functional specifications for the user interface in section 4.1 only some specifications are fulfilled or partly fulfilled. The following lists further indicates whether each of the specifications is satisfied or not.

1. The application runs by entering the port for which the application should run on, and the IP address and port number of the node to connect to. This satisfies the first specification.
2. A website has been created that suggests a way to monitor energy flow, and a way to create smart contracts. The website is not operative, but rather created as a proof-of-concept - partially satisfying this specification.
3. The website login is not implemented, leaving this specification unfulfilled.
4. No functionality for sellers to put up availability was implemented. The website demonstrated in figure 8.5 was hardcoded, but should in reality be updated dynamically. This could for instance be done through a form similar to the smart contract query.
5. A layout for how the buyers can initiate the smart contract with sellers is demonstrated in figure 8.5. However, this is only shown from the buyer's perspective. In order for it to be valid, the contract also needs to be signed by the seller.
6. The ability to monitor the energy flow in real-time is only partially shown. Figure 8.4 suggests how the layout could be, but it is not tested on real-time data. The back-end service of the website needs to store all data for each user. Ideally, this should be taken directly from the blockchain in order to ensure data integrity. However, this might not be feasible as it would require a significant amount of time and computational power to search through the blockchain for all transactions. An alternative approach could be to store electricity data in a separate database.

The user interface requires further work in order to be fully operative. This thesis has merely demonstrated a layout for how this can be achieved, as the user interface was not the main objective in the implementation.

Application

The specifications from the 4.2 section are further evaluated below.

1. As this implementation is only focused around the software, no tests have been done where actual data is received from the smart meters. All tests have been executed based on simulated data stored in a CSV file. However, this solution shows that the system can process the information at periodic intervals.
2. All information about the energy flow in the system is stored in the blockchain, leaving this specification fulfilled.
3. Since the ability to create smart contracts in the user interface was not implemented, the specification of validating and storing smart contracts is not satisfied either. However, the results in section 8.2.4 demonstrate a method for signing and validating smart contracts. This has not been integrated into the system, due to time restrictions. Smart contracts were rather created manually to prove the functionality of the settlement algorithm.
4. Due to a non-operative user interface, the new electricity transactions are not processed on the website.
5. The specification of the settlement process is satisfied. The blockchain leader initiates the settlement every time a new block is validated by the network and added to the blockchain.

Blockchain

The blockchain has been the main focus for the implementation of this system, as this is the fundamental structure that supports the rest of the system. All specifications regarding the blockchain are (partly) fulfilled. As previously mentioned, smart contracts are not integrated into the system, thus affecting the specifications where the smart contracts are involved.

1. The system supports storing electricity transactions in the blockchain, but smart contracts are not supported in the current implementation.
2. The blockchain is distributed between all nodes in the network. The tests show that this can be done with 10 nodes in the system, but should also work for a higher number of nodes.
3. Broadcasting of electricity transactions is implemented. However, due to the same factors as the first specification, the smart contract part of the specification is not implemented.
4. In the current implementation, the leader proposes a new block every time it has transaction data from all nodes in the network for a given interval. As the system is tested on simulated data, new transactions are read in to the system

every two seconds, and new blocks created thereafter. This specification is satisfied, but some modifications needs to be made to the system for it to work on actual data. In reality, a new block will be created every minute, once transaction data for this period is available to the leader.

5. All nodes in the network participate in the consensus process. Further testing on a bigger network needs to be carried out in order to determine if this is the best course of action. An alternative could be to only have a limited number of nodes participate in the consensus process. This is further discussed in section 9.4.
6. The leader of the blockchain keeps track of how many nodes have validated the new proposed block. Once a majority of the network has accepted it, the leader adds it to the blockchain and notifies the other nodes in the network to do the same. Thus, satisfying this specification.
7. As the smart contracts are not stored in the blockchain, the requirement of using them in the settlement process is not satisfied. However, the smart contracts that were manually created, to use in the system test, show that new transactions trigger the settlement based on the smart contracts.
8. The scalability was only tested up to eleven nodes. This proved to have some issues, as the eleventh node, which was started after the others, was not able to be updated on the blockchain. Yet the exact same tests worked for fewer nodes, indicating that there might be some issues with scalability in the consensus protocol.

9.2.2 Evaluation of Related Implementations

Several implementations that already used blockchains for settlement in P2P networks were presented in chapter 3. This section will evaluate what sets the implementation in this thesis apart from these other implementations.

Brooklyn Microgrid

Of the systems presented in chapter 3, the Brooklyn Microgird is the most similar to the system implemented in this thesis. There is no token in the Broolyn Microgrid, the blockchain stores electricity transactions which are later settled. Contrary to the consortium blockchain in the implementation of this thesis, the Brooklyn Microgrid uses a private blockchain. This grants the blockchain owners more control, and the community users of the platform need to trust the blockchain operators. However, the blockchain is more protected as the blockchain owners have complete control over the network. Another important feature of the Brooklyn Microgrid, that separates it from this thesis, is the development of special hardware for monitoring meter readings, as well as running the actual blockchain. This is an elegant solution in regard to limiting the amount of hardware, which is important for a system that is to be installed in people's homes. However, the development of special hardware could be a costly solution.

NAD Grid

The main focus of the NAD Grid lies in their Selective Sourcing technology for reducing carbon footprint. The system is not limited to only microgrids, as the company is partnering with utility companies to label and source electricity from all producers. Furthermore, this sourcing requires tokens to be used. However, the actual electricity transactions are carried out using fiat currency, just like the system in this thesis. The settlement is also done based on smart contracts. Another similarity to the system in this thesis is the use of consortium blockchain for storing all information about electricity exchange.

Power Ledger

The similarities with Power Ledger come from the connection to smart meters, and the use of smart contracts for trading. Meter readings are stored in the blockchain, allowing participants to audit previous usage. However, the Power Ledger platform requires two tokens to work. The advantage of trading tokens directly for consumed power is that settlements occur instantaneously. However, extra interaction with the system is required for purchasing tokens. The additional POWR token is required in order to gain access to the platform. The Power Ledger platform also consists of two blockchains. Like the system implemented in this thesis, a consortium blockchain is used for the actual trading system. Yet, another blockchain is required in the Power Ledger system, a public blockchain, from the Ethereum platform, for handling the token exchange.

A summary of the similarities and differences between the systems is presented in table 9.1.

Table 9.1: Evaluation compared to related systems.

System	**This thesis**	**Brooklyn Microgrid**	**NAD Grid**	**Power Ledger**
Tokens	No	No	Yes	Yes
Blockchain	Consortium	Public to Private	Consortium	Public and Consortium
Smart Contacts	Yes	Yes	Yes	Yes
Automatic Settlement	No	No	Yes	Yes
Microgrids	Yes	Yes	No	Yes
Energy Source	Any	Solar	Any	Solar

9.3 Privacy

Privacy is an important aspect of any digital system, but was not prioritized in this implementation. This section discusses why privacy needs to be included in a fully functioning system, and how this might be achieved.

In order for a system, such as the one proposed in this thesis, to be applied in the real world, the potential users must be assured that their private data is being adequately secured. In blockchains, there is a fine line for balancing transparency and privacy. The transparency is required in order to validate the transactions on the blockchain. However, it is not desirable that just anyone can access user data. Figure 9.1 is taken from the Bitcoin white paper, and shows how blockchains redefine the privacy model. The traditional model where third parties are trusted to handle transactions, hides everything from the public. However, the new privacy model only hides identities from the public, while all transactions are transparent and can be audited by anyone. The implementation in this thesis hides the identities to a certain degree by using an Universally Unique Identifier (UUID) as identification. However, this does not completely protect identities, as IP addresses could be tracked and linked to the identities. The benefit of a consortium blockchain, is that transaction information can be limited to only the members of the network.

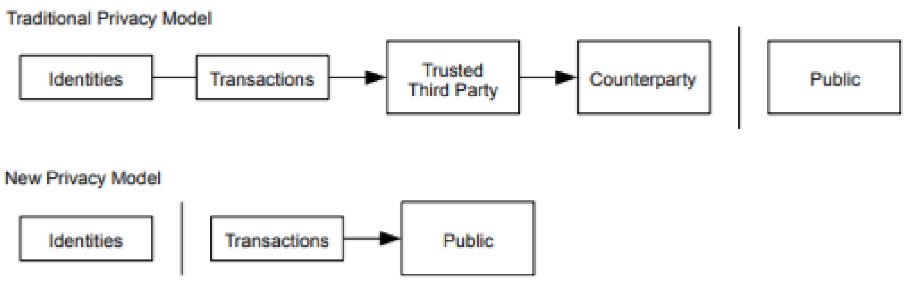


Figure 9.1: Blockchain privacy model [13].

Issues related to privacy may occur if identities are not properly secured, and transaction data is leaked to, or otherwise obtained by, unauthorized parties such as criminals or companies that exploit the data. E.g. criminals could monitor electricity consumption of a household to detect when the house is empty and thus rob it. With all the recent focus concerning data privacy, willingness to use such a system would diminish if transaction data was sold to companies that, for instance, used it to analyze user behavior and thereby using it for marketing and advertising.

One way to ensure that unauthorized people are not given access to data on the blockchain, could be to encrypt all the information that is stored on the blockchain, and only grant access to the decryption key to approved participants of the network. However, the encryption and decryption of data is a fairly heavy computationally operation, and leads to more overhead in the system. This also requires that all members of the network are given the decryption key. This raises issues such as: what would happen if someone accidentally (or purposely) gives away their key information? Such an implementation would require a careful and well tested design.

9.4 Limitations in Implementation

Blockchain technology is still very young, with the first appearance in 2008, in the Bitcoin white paper. The Ethereum platform, launched in 2015, is one of the very first to use the blockchain technology for something else than just crypto currency. In the past couple of years, there have been huge advantages in the field, and many new blockchain applications have been created. Furthermore, several of the papers used for the background research of this thesis have been published within the past year.

Among the newer blockchain technologies is the Hyperledger Fabric platform. This platform has great potential for developing a system such as the one presented in this thesis. However, rather than spending a lot of time learning a specific framework, the goal was rather to learn the blockchain functionality, and this was done by implementing a blockchain from scratch. In retrospect, this might not have resulted in the best possible system. However, it has given tremendous insight in how blockchains work and their abilities - and limitations.

The advantages of building the blockchain structure from scratch is that it can be formed exactly how the system specifications require it to be. Furthermore, the system can be implemented without transaction fees, which would be required e.g. on the Ethereum platform.

The disadvantages of not utilizing existing blockchain platforms is that it requires more extensive testing to ensure correct behavior, more maintenance, and more documentation. By using an existing blockchain platform, like Hyperledger Fabric, the development of the system would most likely be faster, seeing as the low-level modules are implemented, thoroughly tested, and ready for use. Hyperledger also has a built-in smart contract program in *chaincode*.

Consensus

As previously mentioned, the consensus module is not without flaws, as became clear in the testing.

Other issues regarding the module include the lack of protection against Byzantine nodes. It can be argued that in a consortium blockchain where nodes are validated, fault-tolerance is sufficient and the system does not need to be BFT. Furthermore, as such a system would be restricted to a limited physical area, it is quite likely that network participants are acquaintances in real life, thus reducing the risk of Byzantine nodes. Nonetheless, in the current implementation new network participants are not part of the validation process, they simply require the IP address and port number of a node in the network. A node conducting a brute force attack could quite easily access the network and establish itself as a leader, simply by sending out *append entries* messages.

A method of improving the consensus protocol could be to limit the number of consensus nodes to e.g. 5 nodes. The process of selecting which nodes to participate in the consensus process could be implemented in a similar way to how leaders are elected. This would result in a more centralized system, but could improve scalability and performance. However, this would require and alternative method of ensuring consistent *proposed blocks* logs among the followers, as the leader would no longer have access to update the logs of all it's followers.

One of the big advantages of the Raft protocol, apart from being very understandable and quite easy to implement, is that there are no possibilities for forks in the network. In Bitcoin, and other blockchains with PoW consensus, transactions are never finalized, in the sense that they can become part of a fork. As time passes and more blocks are created, a transaction is less likely to be part of a fork. In Raft however, transactions are finalized when the block becomes part of the blockchain. Due to the leader configuration and validation prior to the blockchain, forks do not occur. Once a block is committed to the blockchain, it is committed by all nodes and no other block can be committed in its place, due to the strong consistency of the consensus protocol. Another advantage, is that there is no need for any tokens or crypto-currencies in the network, contrary to protocols like PoW and PoS. In contrast to the PoW protocol in the Bitcoin blockchain, which require enormous amounts energy to operate, the Raft protocol is a lightweight protocol, that can run on normal processing capacity.

No previous blockchain implementations of Raft could be found during the background research. This thesis shows that the consensus protocol can be modified to work in a blockchain network.

Network

In the current implementation of the network module, all nodes have an open connection to all other nodes in the system. This solution does not scale well when the network is extended. One possibility to solve this problem could be to limit each node to e.g. 5 connections. This requires that all messages are broadcasted and forwarded until they have reached all nodes in the network, and could be achieved by numbering all messages.

The tests show the network protocol works very well apart from being unable to operate through different WiFi networks. The *Twisted* library eased the development of the network communication, as the low-level components of the protocol were abstracted away. This allowed for a well-structured and modularized API.

Storage

As shown in the results, a block containing energy transactions from 4 nodes resulted in approximately 423 bytes. With a new block created every minute, this will eventually produce a large sized blockchain. A solution to prevent this, could be to limit the blockchain horizon to 3 years, which is how long power companies are required to store readings [67]. Thus, after year 4, the first year worth of blocks will be removed from the blockchain and the genesis block rewritten to match the new first block of the blockchain. Additionally, when the smart contract feature is fully implemented and the contracts are stored in their own blockchain, contracts that are no longer valid should be removed from the blockchain after a certain period.

9.5 Future Work

A discussion of features and elements that should be included in a real system is presented in this section. A summary of all future work is listed at the end of the section.

The most important aspect missing in the system is the smart contract API. Considering that the settlement protocol builds on the smart contract API, this feature must be implemented in order for the system to operate. This API is also important in order for the users of the system to interact with each other. Furthermore, a method of creating smart contracts automatically, based on supply and demand, and preset conditions in the system, could be an interesting feature to implement. Based on pricing and availability, the program should be able to match buyers and sellers of electricity in an optimal manner.

Settlements in energy systems are quite complex. The implementation of such an algorithm requires extensive knowledge on power grids to produce an optimal algorithm. This was out of the scope for this thesis, but is something that needs to be implemented in a real system.

The system has only been tested on simulated data. The system needs to be implemented on actual hardware, including smart meters, and tested with actual user data.

Following is a summary of the future work that should be carried out on the system, based on what was discussed in this section and in section 9.4.

- Improve privacy
- Further assessment of consensus module, perhaps a better alternative than Raft.
- Limit number of open connections in network protocol
- Enable network communication across different networks

- Storage optimization
- Optimization of settlement algorithm
- Connection to smart meters
- Full integration of smart contracts
- Continued development of user interface
- Automatic creation of smart contracts

Chapter 10

Conclusions

The main goal of this thesis was to investigate how blockchains could be used to settle electricity transactions in a microgrid. First, extensive background research on the subject was done. Further, existing systems using blockchains for electricity trading in P2P markets were reviewed. Based on the findings in the background research, a new blockchain system capable of storing electricity transactions was proposed and implemented. Lastly, a method of settling these transactions was proposed and implemented, on top of the developed blockchain infrastructure.

The results from the tests show that the implemented system is capable of storing electricity transactions in an immutable ledger, and settling these transactions between parties of a smart contract. Thus, proving that blockchains can be used as the back-end for a system of settlement in a microgrid. However, the implemented system is not without flaws, nor is it exhaustive.

The main contribution from this thesis is a blockchain built from scratch, using Raft as the consensus protocol. The implemented blockchain does not use any crypto-currencies or tokens, which separates it from other similar systems. Furthermore, the energy sources in the microgrid are not limited to solar power, rather any type of energy source may be connected to the grid. Blockchains can be used for any types of transactions, financial or not. In the system implemented in this thesis, the benefits from using a blockchain include that no two nodes can claim the same financial payment for the same unit of energy.

The blockchain technology is still a fairly new technology with new advances occurring at a rapid speed. However, blockchains have already shown great potential in many different application areas, including in energy trading systems, as showed in this thesis.

Appendices

Appendix A

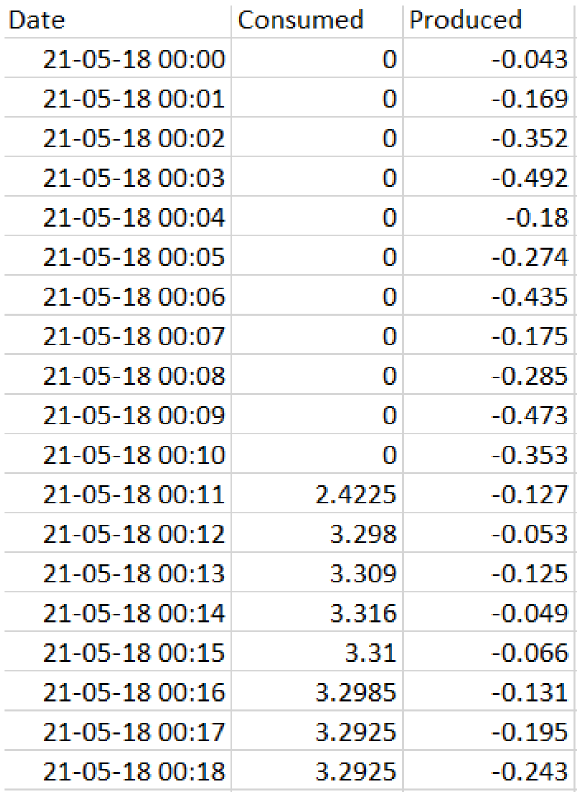
Digital Signatures

Signing and verifying a message using private and public keys from the ECDSA library.

```
from ecdsa import SigningKey
import uuid

node1 = str(uuid.uuid4())
node2 = str(uuid.uuid4())

private_key_1 = SigningKey.generate()
private_key_2 = SigningKey.generate()

public_key_1 = private_key_1.get_verifying_key()
public_key_2 = private_key_2.get_verifying_key()

contract = "Test"

signature1 = private_key_1.sign(contract.encode('utf-8'))
signature2 = private_key_2.sign(contract.encode('utf-8'))

print(public_key_1.verify(signature1, contract.encode('utf-8')))
print(public_key_2.verify(signature2, contract.encode('utf-8')))

print(public_key_2.verify(signature1, contract.encode('utf-8')))
```

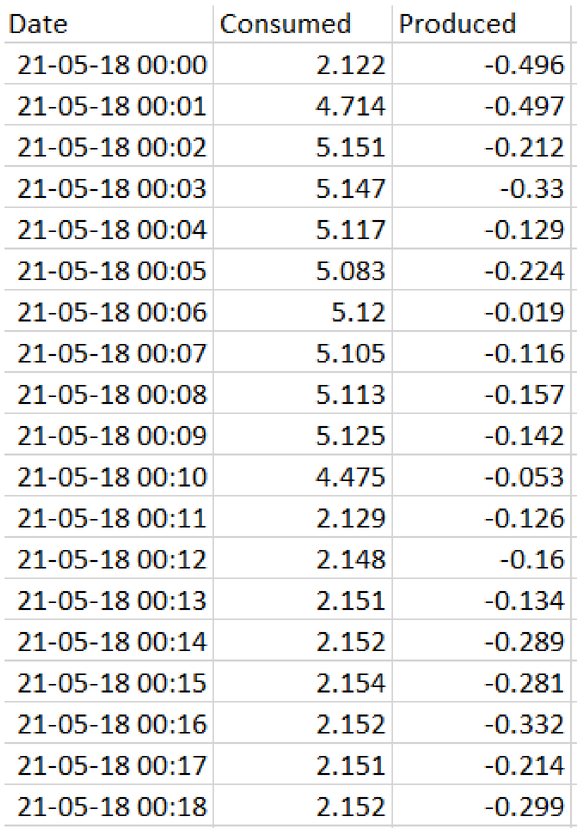
Appendix B

Test Data



Date	Consumed	Produced
21-05-18 00:00	0	-0.043
21-05-18 00:01	0	-0.169
21-05-18 00:02	0	-0.352
21-05-18 00:03	0	-0.492
21-05-18 00:04	0	-0.18
21-05-18 00:05	0	-0.274
21-05-18 00:06	0	-0.435
21-05-18 00:07	0	-0.175
21-05-18 00:08	0	-0.285
21-05-18 00:09	0	-0.473
21-05-18 00:10	0	-0.353
21-05-18 00:11	2.4225	-0.127
21-05-18 00:12	3.298	-0.053
21-05-18 00:13	3.309	-0.125
21-05-18 00:14	3.316	-0.049
21-05-18 00:15	3.31	-0.066
21-05-18 00:16	3.2985	-0.131
21-05-18 00:17	3.2925	-0.195
21-05-18 00:18	3.2925	-0.243

Figure B.1: Transaction data for node 274b2397-508c-419d-aa75-d0a909a80dd1



Date	Consumed	Produced
21-05-18 00:00	2.122	-0.496
21-05-18 00:01	4.714	-0.497
21-05-18 00:02	5.151	-0.212
21-05-18 00:03	5.147	-0.33
21-05-18 00:04	5.117	-0.129
21-05-18 00:05	5.083	-0.224
21-05-18 00:06	5.12	-0.019
21-05-18 00:07	5.105	-0.116
21-05-18 00:08	5.113	-0.157
21-05-18 00:09	5.125	-0.142
21-05-18 00:10	4.475	-0.053
21-05-18 00:11	2.129	-0.126
21-05-18 00:12	2.148	-0.16
21-05-18 00:13	2.151	-0.134
21-05-18 00:14	2.152	-0.289
21-05-18 00:15	2.154	-0.281
21-05-18 00:16	2.152	-0.332
21-05-18 00:17	2.151	-0.214
21-05-18 00:18	2.152	-0.299

Figure B.2: Transaction data for node 712b259d-3d24-431e-b4b8-e091b3a38ad6

Bibliography

[1] Investopedia Staff. “The collpas of Lehman Brorthers: A case study”. In: *Investioedia* (Dec. 2017). URL: https://www.investopedia.com/articles/economics/09/lehman-brothers-collapse.asp.

[2] Satoshi Nakamoto. *Bitcoin open source implementation of P2P currency.* URL: http://p2pfoundation.ning.com/forum/topics/bitcoin-open-source.

[3] Marco Iansiti and Karim R. Lakhani. “The Truth About Blockchain”. In: *Harvard Business Review* (Jan. 2017). URL: https://hbr.org/2017/01/the-truth-about-blockchain.

[4] M. Mihaylov et al. “NRGcoin: Virtual currency for trading of renewable energy in smart grids”. In: *11th International Conference on the European Energy Market (EEM14).* 2014, pp. 1–6. DOI: 10.1109/EEM.2014.6861213.

[5] *Brooklyn Microgrid 101.* URL: http://brooklynmicrogrid.com/.

[6] Ernie Hayden. “Introduction to Microgrids”. In: (). URL: https://www.securicon.com/sites/default/files/Introduction%20to%20Microgrids%20-%20Securicon%20-%202013_1.pdf.

[7] *Microgrids, the self-healing solution.* URL: https://www.generalmicrogrids.com/about-microgrids.

[8] O.I. Konashevych. “Advantages and Current Issues of Blockchain Use in Microgrids”. In: *Electronic Modeling - Vol. 38, No. 2.* 2016, pp. 93–103. URL: http://dspace.nbuv.gov.ua/handle/123456789/101347.

[9] *How Microgrids Helped Weather Hurrican Sandy.* URL: https://www.greentechmedia.com/articles/read/how-microgrids-helped-weather-hurricane-sandy#gs.gO_LP24.

[10] ABB. *Introduction to Microgrids.* URL: http://new.abb.com/distributed-energy-microgrids/introduction-to-microgrids.

[11] Mary M Timney. *Power for the People: Protecting States’ Energy Policy Interests in an Era of Deregulation: Protecting States’ Energy Policy Interests in an Era of Deregulation.* Routledge, 2015. ISBN: 9781317462293.

[12] Janusz Bialek. “Tracing the flow of electricity”. In: *IEE Proceedings-Generation, Transmission and Distribution* 143.4 (1996), pp. 313–320.

[13] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system.* 2009. URL: `http://bitcoin.org/bitcoin.pdf`.

[14] Michael Nofer et al. "Blockchain". In: *Business & Information Systems Engineering* 59.3 (June 2017), pp. 183–187. ISSN: 1867-0202. DOI: `10.1007/s12599-017-0467-3`.

[15] Vitalik Buterin. *On Public and Private Blockchains.* Blog. Aug. 2015. URL: `https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/`.

[16] X. Xu et al. "A Taxonomy of Blockchain-Based Systems for Architecture Design". In: *2017 IEEE International Conference on Software Architecture (ICSA).* Apr. 2017, pp. 243–252. DOI: `10.1109/ICSA.2017.33`.

[17] Z. Zheng et al. "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends". In: *2017 IEEE International Congress on Big Data (BigData Congress).* June 2017, pp. 557–564.

[18] Wenting Li et al. "Securing Proof-of-Stake Blockchain Protocols". In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology.* Ed. by Joaquin Garcia-Alfaro et al. Cham: Springer International Publishing, 2017, pp. 297–315. ISBN: 978-3-319-67816-0.

[19] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine Generals Problem". In: *ACM Trans. Program. Lang. Syst.* 4.3 (July 1982). ISSN: 0164-0925. DOI: `10.1145/357172.357176`. URL: `http://doi.acm.org/10.1145/357172.357176`.

[20] N. Zhumabekuly Aitzhan and D. Svetinovic. "Security and Privacy in Decentralized Energy Trading through Multi-signatures, Blockchain and Anonymous Messaging Streams". In: *IEEE Transactions on Dependable and Secure Computing* 99 (Oct. 2016), pp. 1–14. ISSN: 1545-5971. DOI: `10.1109/TDSC.2016.2616861`.

[21] Scott Driscoll. *How Bitcoin works under the hood.* URL: `http://www.imponderablethings.com/2013/07/how-bitcoin-works-under-hood.html`.

[22] K.J. O'Dwyer. "Bitcoin Mining and its Energy Footprint". In: *IET Conference Proceedings* (Jan. 2014), 280–285(5). URL: `http://digital-library.theiet.org/content/conferences/10.1049/cp.2014.0699`.

[23] *Bitcoin Energy Consumption Index.* URL: `https://digiconomist.net/bitcoin-energy-consumption`.

[24] Dr. Arati Baliga. *Understanding Blockchain Consensus Modles.* Tech. rep. Apr. 2017. URL: `https://www.persistent.com/wp-content/uploads/2017/04/WP-Understanding-Blockchain-Consensus-Models.pdf`.

[25] *Ethereum wiki - Proof of Stake FAQ.* URL: `https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ`.

[26] *Proof of Stake - The Rich get Richer...Less!* 2018. URL: `https://steemit.com/crypto/@neoversion6/proof-of-stake-the-rich-get-richer-less`.

[27] Gert Rammeloo. "The economics of the Proof of Stake consensus algorithm - Proof of Stake: an economic analysis". In: *Meduim* (2017). URL: `https:`

//medium.com/@gertrammeloo/the-economics-of-the-proof-of-stake-consensus-algorithm-e28adf63e9db.

[28] Diego Ongaro and John Ousterhout. "In Search of an Understandable Consensus Algorithm". In: *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference.* USENIX Association, 2014, pp. 305–320. ISBN: 978-1-931971-10-2. URL: http://dl.acm.org/citation.cfm.

[29] Leslie Lamport. "Paxos Made Simple". In: 2001.

[30] Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance and Proactive Recovery". In: *ACM Trans. Comput. Syst.* 20.4 (Nov. 2002), pp. 398–461. ISSN: 0734-2071. DOI: 10.1145/571637.571640. URL: http://doi.acm.org/10.1145/571637.571640.

[31] Amrutha Gayathri. "From marijuana to LSD, now illegal drugd delivered on your doorstep". In: *International Business Times* (June 2011).

[32] Vitalik Buterin. *Visions Part 1: The Value of Blochain Technology.* Blog. Apr. 2015. URL: https://blog.ethereum.org/2015/04/13/visions-part-1-the-value-of-blockchain-technology/.

[33] Alex Lielacher. *Sending Money? Why Not Do it with Bitcoin.* Dec. 2017. URL: https://wirexapp.com/use-bitcoin-send-receive-international-money-transfers/.

[34] World Bank Group. "Remittance Prices Worldwide". In: (Mar. 2017). URL: https://remittanceprices.worldbank.org/sites/default/files/rpw_report_march_2017.pdf.

[35] Steven Buchko. *How Long do Bitcoin TransactionsTake?* Dec. 2017. URL: https://coincentral.com/how-long-do-bitcoin-transfers-take/.

[36] Stan Higgins. "Bitcoin-Powered Crowdfunding App Lighthouse Has Launched". In: *Coindesk* (2015). URL: https://www.coindesk.com/bitcoin-powered-crowdfunding-app-lighthouse-launches-open-beta/.

[37] *Ethereum - Blockchain App Platform.* URL: https://www.ethereum.org/.

[38] Alyssa Hertig. "Who created Ethereum". In: *Conidesk* (Mar. 2017). URL: https://www.coindesk.com/information/who-created-ethereum/.

[39] Vitalik Buterin. *Ethereum White Paper.* Tech. rep. Ethereum Foundation, 2013. URL: https://github.com/ethereum/wiki/wiki/White-Paper.

[40] *What is Ethereum?* URL: http://www.ethdocs.org/en/latest/introduction/what-is-ethereum.html.

[41] *News by the people, for the people.* URL: https://dnn.media/.

[42] *Open Identity System for the Decentralized Web.* URL: https://uport.me.

[43] Linux Foundation. *About Hyperledger.* URL: https://www.hyperledger.org/about.

[44] Linux Foundation. *Hyperledger Fabric.*

[45] Linux Foundation. *Industries.* URL: https://www.hyperledger.org/industries.

[46] *SWIFT's Blockchain Pilot For Bank-To-Bank Transfers Went Extremely Well.*

[47] Maciek Jedrzejczyk. *eVoting in Poland becomes a moonshot with blockchain.* Jan. 2018. URL: https://www.linkedin.com/pulse/evoting-

poland-becomes-moonshot-blockchain-maciek-j%C4%99drzejczyk/?trackingId=hIoC1Jyjj0c58Ts6MwpEMA%3D%3Dtd%3E.

[48] *A peer-to-peer platform for efficient trade of electricity*. URL: https://www.nadgrid.com/.

[49] Sujha Sundarajan. "Japanese Shipping Giant, IBM to Trial Blockchain in Cross Border Trade". In: *Coindesk* (Dec. 2017). URL: https://www.coindesk.com/japanese-shipping-giant-ibm-to-trial-blockchain-in-cross-border-trade/.

[50] Claude R. Olsen. "Blokkjeder år fart på verdenshandelen". In: *Teknisk Ukebland* (Apr. 2018), pp. 76–81. ISSN: 0040-2354.

[51] Nick Szabo. "Formalizing and Securing Relationships on Public Networks". In: *First Monday* 2.9 (Sept. 1997). ISSN: 13960466.

[52] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. "A survey of attacks on Ethereum smart contracts (SoK)". In: *International Conference on Principles of Security and Trust*. Springer. 2017, pp. 164–186.

[53] *Electric Power Technical Whitpaper*. Tech. rep. Dec. 2017. URL: https://exergy.energy/wp-content/uploads/2017/12/Exergy-Whitepaper-v8.pdf.

[54] Clinton Nguyen. "Brooklyn's 'Microgrid' Did Its First Solar Energy Sale". In: *Motherboard, Vice* (Apr. 2016). URL: https://motherboard.vice.com/en_us/article/d7y7n7/transactive-grid-ethereum-brooklyn-microgrid.

[55] Urszula Papajak. "Can the Brooklyn Microgrid project revolutionize the energy marke?" In: *Medium* (Nov. 2017). URL: https://medium.com/thebeammagazine/can-the-brooklyn-microgrid-project-revolutionise-the-energy-market-ae2c13ec0341.

[56] James Basden and Michael Cottrell. "How Utilities Are Using Blockchain to Modernize the Grid". In: *Hardware Business Review* (2017). URL: https://hbr.org/2017/03/how-utilities-are-using-blockchain-to-modernize-the-grid.

[57] Esther Mengelkamp et al. "Designing microgrid energy markets: A case study: The Brooklyn Microgrid". In: *Applied Energy* 210 (2017), pp. 870 –880. ISSN: 0306-2619. DOI: https://doi.org/10.1016/j.apenergy.2017.06.054. URL: http://www.sciencedirect.com/science/article/pii/S030626191730805X.

[58] *How NAD Grid Work*. URL: https://nadgrid.com/.

[59] Coinstar. *NAD Grid - Thomas Sun - APAC Blockchain Conference 2018*. Youtube. Mar. 2018. URL: https://www.youtube.com/watch?v=euTqN3ZJE0O.

[60] *Power Ledger*. URL: https://powerledger.io.

[61] *Power Ledger White Paper*. Tech. rep. 2018. URL: https://powerledger.io/media/Power-Ledger-Whitepaper-v8.pdf.

[62] *Python Software Foundation*. URL: https://www.python.org/.

[63] *Twisted Matrix Labs*. URL: https://twistedmatrix.com/trac/.

[64] *Flask - Web Development One Drop at a Time*. URL: http://flask.pocoo.org/docs/1.0/ (visited on 05/25/2018).

[65] *Pure-Python ECDSA*. URL: `https://github.com/warner/python-ecdsa` (visited on 05/25/2018).

[66] *hashlib - Secure hashes and message digests*. URL: `https://docs.python.org/3/library/hashlib.html`.

[67] Øyvind Lie Leif Hamnes. "Vil lagre dine strømdta i 10 år". In: *Teknisk Ukeblad* (July 2012). URL: `https://www.tu.no/artikler/vil-lagre-dine-stromdata-i-ti-ar/235540` (visited on 05/30/2018).