Norwegian University of Science and Technology

Department of Engineering Cybernetics

Utilizing Blockchain Technology for Settlement in a Microgrid

By:
Marit Schei Tundal

Supervisor:
Geir Mathisen

May, 2018

Problem Description

In a limited electrical system (a microgrid, such as an island community) there will be some, 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 storage 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 task will be:

- Look at 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

The problem description for this thesis was given by supervisor Geir Mathisen.

Abstract

Sammendrag

Abbreviations

RPC	=	Remote Procedure Call
EVM	=	Ethereum Virtual Machine
API	=	Application Programming Interface
BFT	=	Byzantine Fault-Tolerance
BGP	=	Byzantine Generals Problem
PoW	=	Proof of Work
PoS	=	Proof of Stake
ASIC	=	Application Specific Integrated Circuits
PBFT	=	Practical Byzantine Fault Tolerance
PoC	=	Proof of Concept
P2P	=	Peer-to-Peer
NOK	=	Norwegian Kroner

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 them, the investment bank, Lehman Brothers, filed for bankruptcy [1]. The events that followed launched a global financial crisis with the stock market dropping and unemployment rate increasing world wide. The banks who initially caused the crises, however, were bailed out with tax payer money. People no longer felt they could trust bankers or investment managers. As a response to this, Satoshi Nakamoto outlined a peer-to-peer electronic cash system in the Bitcoin whitepaper.

When the bitcoin system became a reality in 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 traditional trusted middlemen and third-parties. The Bitcoin technology removed the need for trust in transactions, due to the distributed, decentralized ledger where

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

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, blockchains 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 has been massive development with many new applications created, 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.

Among the industries who have adopted the blockchain as a method to improve transactions, is the electricity market. 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.

1.2 Limitations

In a complete system, there are several aspects that ought to be considered. Among these are privacy,... The system implementation in this thesis, however, is focused around the technical design, and therefor these elements are not part of the implementation. Details around these aspects will be further discussed in chapter 9

1.3 Contribution

The blockchain technology is still fairly new. Contributions in this thesis are summarized as:

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

1. Outline existing implementations of blockchains used in peer-to-peer electricity trading.
2. Propose a new blockchain-like system for storing electricity transactions.
3. Propose a 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. Related work on projects using blockchains in energy transactions are presented in chapter 3.

The functional specifications of the system to be implemented are presented in chapter 4. The system mainly consists of two parts: implementation of a blockchain for storing data in a distributed network; and an application for settlement of energy transactions in a microgird. The system implementation is focused around the blockchain. The blockchain is the back-end service enabling the functionality of the application.

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 chapter 7 and 8, respectively. Discussion and conclusion of the thesis will be presented in chapter 9 and 10.

Chapter 2

Background

This chapter mainly contains relevant background theory on blockchains. The chapter starts off with a very basic introduction to microgrids. Related work on the subject is presented at the end of the chapter.

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. A formal definition of a microgrid is as follows:

"*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.*" [6]

Microgrids usually include renewable energy sources, such as wind generators, small hydro power generation, and photovoltaic solar panels, that are locally grouped together. Microgrids may also include generators and batteries to ensure reliability [7]. By placing power generation and power 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 getting power. This can typically be rural places or islands. They 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].

INCLUDE SOME STUFF ON HOW ELECTRONS TRAVEL ON THE GRID

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 [11].

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 [12]. Due to the decentralization of the network, there is no single-point-of-failure, as the database 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[11]. 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 the 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. The recipient node sends the hash of its public key to the node initiating the transaction. The public key, together with the message is hashed together, and used to create a digital signature, thus providing a unique signature for every transaction. The transaction can then be claimed by the node in possession of the corresponding

private key. Thus, in order to use a previously obtained output as an input, it must be verified by the node's private key. 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. Since every transaction is visible to every node on the network, anonymity is important as it becomes (nearly) [13] impossible to find the true identity of the owner. A new key pair can be generated for every transaction to increase anonymity.

A miner stores multiple pending and unconfirmed transaction together to create a block. 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 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 miners find the solution to the next block at the exact same time, a fork is created. 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 [11]. 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. It is therefore a good policy to wait until the block containing the transactions has multiple successors before asserting the transaction 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 [14, 15, 16].

- Public blockchains, like Bitcoin, is a blockchain 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 the two, 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*"[14].

Authorization in the blockchain depends on whether they are permissioned or permissionless. Permissionless blockchains are open to anyone, while permissioned blockchains require approval from the owner or 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.

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

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.

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 [17]. There are two different types of fault-tolerance in consensus mechanisms. 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) [18]. 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.

2.4.1 Proof of Work

Proof of Work (PoW) is the consensus model presented by Nakamoto in the Bitcoin white paper [11], and the most commonly used consensus protocol in blockchains CITE. 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. The required work consists of solving a computationally expensive cryptographic problem. Each new block has a unique hash based. The problem consists of combining this hash with a variable called nonce, to find a hash that is less than a given target (e.g. a 256 bit hash containing x significant zeros). 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 2.1

Nodes working on fining these proofs are called miners, and receive some form of rewards 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 [20]. By letting anyone participate as a miner it is ensured that the network has complete decentralization.

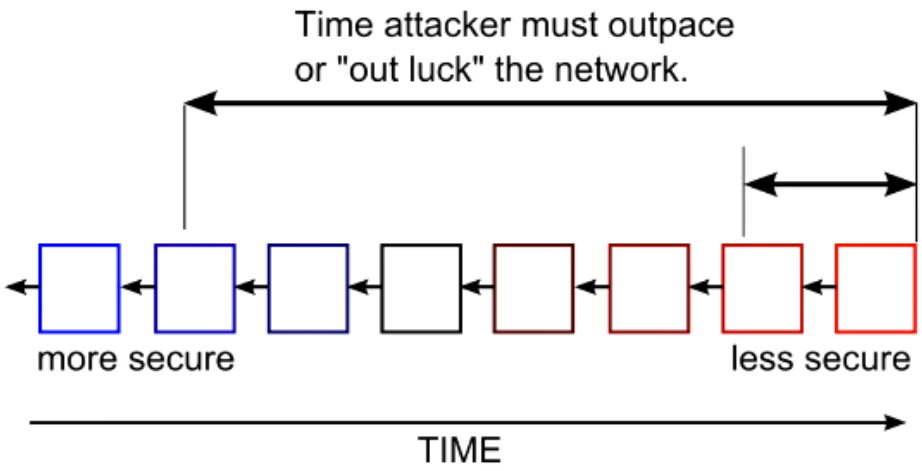

Figure 2.1: Transactions in blocks are in less danger of being altered as more time passes [19]

As described by Nakamoto, PoW is also a method for ensuring that a majority always controls the blockchain [11]: "*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 consumed more energy than the entire country of Switzerland [21], as seen in figure 2.2.

Nakamoto shows that the probability of an attacker altering a block decreases as more blocks are added on top of it [11]. 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.

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, compared to VISAs throughput of 10 000 transactions per second [22].

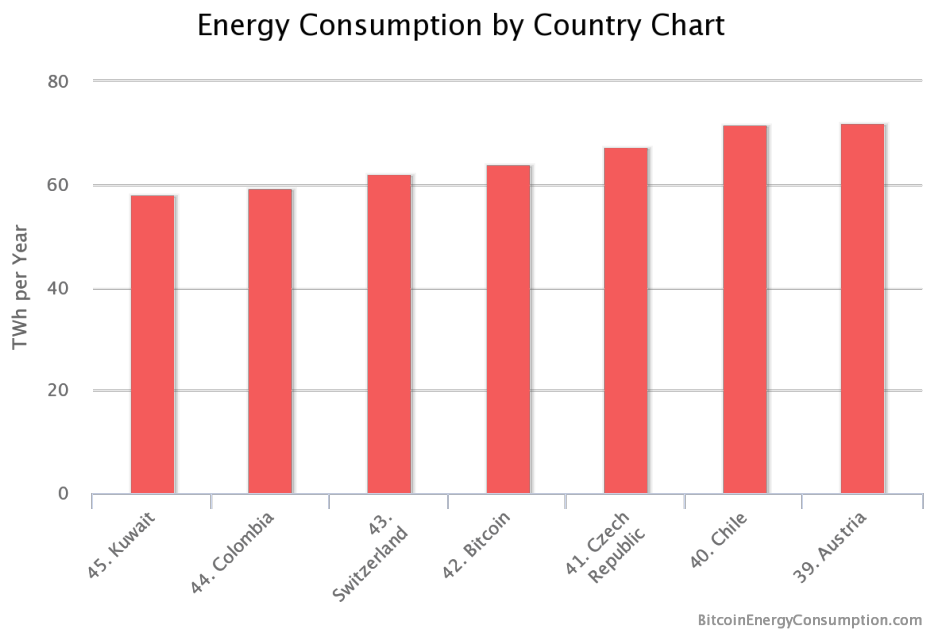

Figure 2.2: Bitcoins energy consumption index relative to countries in the world [21]

2.4.2 Proof of Stake

Proof of Stake (PoS) was created to solve the energy problem of PoW [22, 16]. 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 [23] 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 validated 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 [22, 17]. This problem is known as the nothing-at-stake problem. Ethereum, who 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 [23].

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 [24] that PoW favors the rich even more, as it is more expensive to purchase hardware required for mining and electricity costs, than buying coins in the blockchain to be more likely to participate in the validation process [25].

2.4.3 Raft

Raft is a partially asynchronous, leader based consensus algorithm for managing replicated logs across a distributed system [26]. Raft was designed to be a more understandable alternative to Paxos [27]. The algorithm was first proposed by Onargo et al [26]. 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.

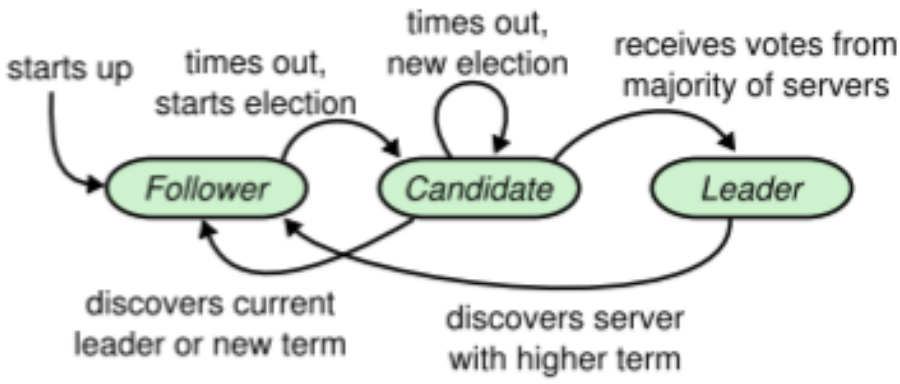

Figure 2.3: Server states. [26]

Time is divided into terms in Raft. Each term is of arbitrary length, depending on the operation of the leader. Each term starts with a **leader election**. If a 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:

- "**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 server 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. 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.

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.

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:

- 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. [18]. However, the solutions presented in the paper are expensive in regard to time and message overhead, thus making it impractical to use in real systems. A solution to this problem was proposed by Castro et.al [28] 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 2.4.

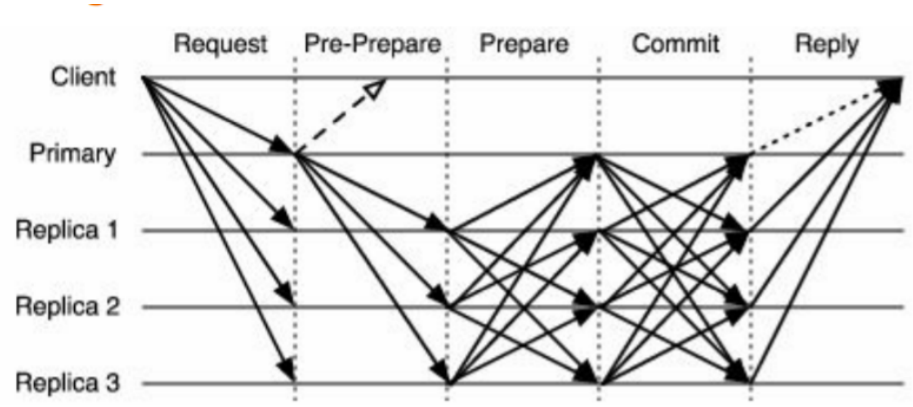

Figure 2.4: Three-phase protocol for consensus in PBFT [28].

Hyperledger Fabric provides consensus as a modular component in the system, supporting various kinds of consensus models. One module utilizes PBFT to order transactions in the system, thus ensuring that all peers have the same list of transactions in their ledger [29, 30].

2.4.5 Summery 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		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. These challenges will be further discussed in the following section.

2.5 Blockchains and Their Applications

Numerous blockchains have been created since the Bitcoin blockchain was published. Many blockchains, like Bitcoin, are based around crypto-currency. However, there are several blockchain 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 for. 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 [32].

Another major advantage of Bitcoin is 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 in Bitcoin to projects.

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.

A smart contract is code enabling self-enforcing applications, through a pre-defined set of rules in the code. The smart contract code is stored in blocks which in turn become part of the blockchain. The smart contracts are used to establish rules between peers. A new programming language, solidity [40], was developed for creating smart contracts.

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

Hyperledger consists of over 100 members, including companies such as IBM, Intel, Samsung, J.P. Morgan, American Express and Airbus [29]. 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.

The Hyperledger blockchains differ from each other, but all have in common that they do not have any 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.*" [29]

The Hyperledger Fabric version 1.0 was released in July 2017, and the new version Fabric v1.1 was realeased in March 2018. The blockchain is a base for developing blockchain with modular architecture [43], and aims to be a plug-and-play solution for various applications. 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 [44]. 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*"

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

- Bank-to-Bank transfers [45]
- eVoting [46]
- Peer-to-peer energy trading [47]
- Cross-border trade operations [48]

In 2016, transport company Maersk partnered with IBM to create a platform where blockchains are used in the cross-border shipping processes [49]. To 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 an API, 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 [11, 40, 29]

Blockchain	**Bitcoin**	**Ethereum**	**Hyperledger Fabric**
Founded	2009	2015	2017
Crypto currency	Bitcoin	Ether	Non
Consensus	PoW	PoW, PoS	PBFT
Permissioned	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

AVSNITT IKKE FULLFØRT Nick Szabo 1996 Constant read-only contracts, or transactional contracts A smart contract is a software program that adds layers of information onto digital transactions being executed on a blockchain. It allows for more complex transactions than simply exchanging digital tokens for a product or service.

Third party - executed based on events

vending machine metaphor

A smart contract can only be as smart as the people coding taking into account all available information at the time of coding. While smart contracts have the potential to become legal contracts if certain conditions are met, they should not be confused with legal contracts accepted by courts and or law enforcement. However, we will probably see a fusion of legal contracts and smart contracts emerge over the next few years as the technology becomes more mature and widespread and legal standards are adopted.

Inherit properties of the blockchain - immutable and distributed

1. Pre-defined contract: terms established by counter-parties, conditions for execution
2. Events: trigger contract execution, e.g. transaction initiated or information received
3. Execute and value transfer: terms of contract dictate movement of value based on conditions met
4. Settlement: digital assets on chain are automatically settled; physical assets - changes to accounts on the ledger will match off-chain settlement instructions

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 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 [5]. This is done in a peer-to-peer 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 [50, 51] the first ever energy transaction made on a blockchain.

The company started working with the public blockchain Ethereum [52, 51], which was the first blockchain to make a non-financial transaction. However, they quickly realized that this was not a feasible solution, 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 [52, 53]. 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. [54] 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 gird in case of power outages. The virtual platform is the technical infrastructure, and runs a private blockchain. The virtual layer is where all the information of the network is transferred.

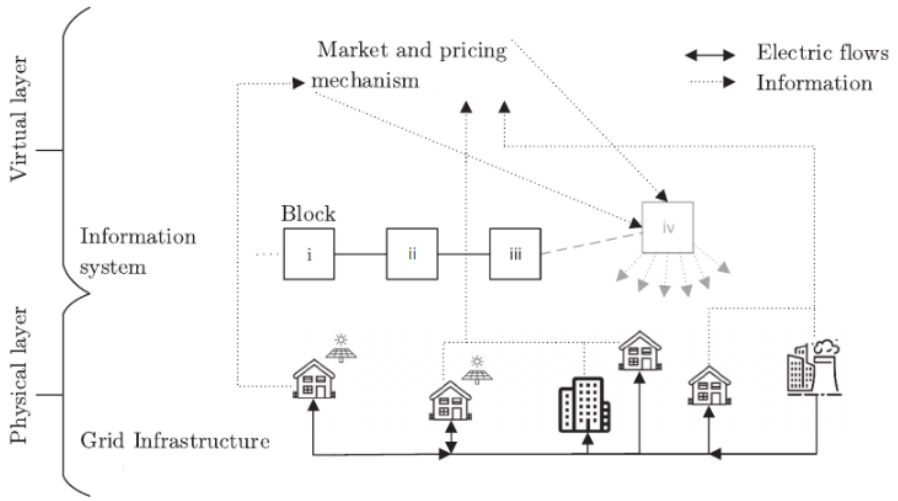

Figure 3.1: High-level topology of the Brooklyn Microgrid [54].

3.2 NAD Grid

NAD Grid is a platform for peer-to-peer energy transactions over the blockchain, based in Silicon Valley [55]. The company focuses on reducing carbon footprint through their Selective Sourcing technology [56].

The software platform is built upon the physical grid, and they aim to partner with existing utility companies [55], who own the grid, to provide a platform that allow users to trade electricity in real-time over the NAD Exchange [56]. As of March 2018 [55], 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 [47], 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.

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. Consumers can specify a carbon footprint grade they want, and 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 [57]. 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 [58].

The Power Ledger platform consists of two blockchains, each with its own token:

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.

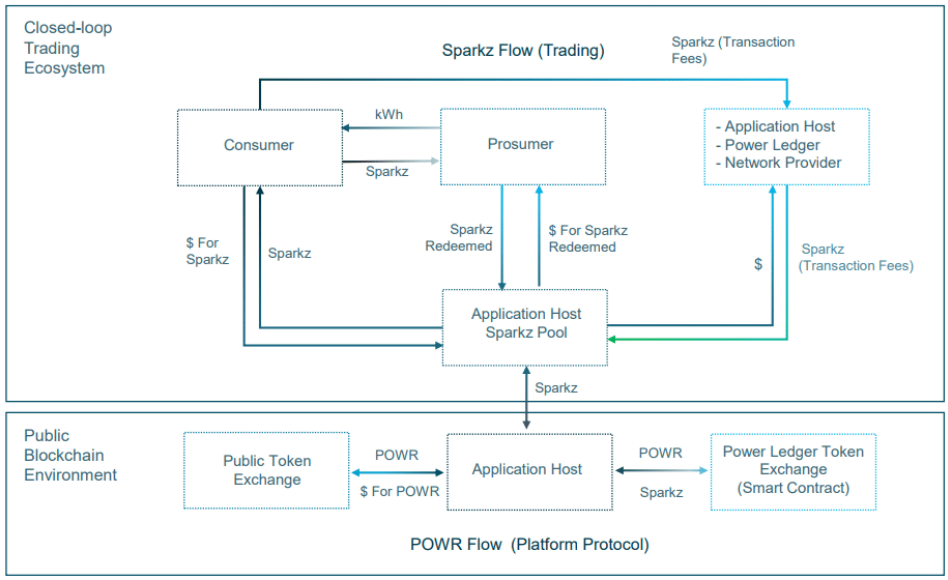

Figure 3.2: Power Ledger Platform [57].

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: application from the user's perspective; application from computer's perspective; 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 is a costly solution, and should be avoided. Participants can be discouraged from this solution by setting extreme prices on electricity from these sources.

4.1 Application From User Perspective

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 monitor production and consumption of electricity.
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 Use Case

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, which is the node ID and a corresponding password. 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. 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 From Computer Perspective

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 Use Case

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

4.3.1 Use Case

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.

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 blockchain layer and the application layer design are both illustrated in this section. The interaction between the layers is illustrated at the end of the section. FIGURES

5.1.1 Blockchain Layer

5.1.2 Application Layer

5.1.3 Interaction Between Layers

5.2 Module Design

The individual modules in each of the are explained in further detail in this section.

5.2.1 Smart Contract

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 prosumer and consumer.
4. The minimum amount of electricity transacted between the contract parties.
5. Signatures from both parties of the contract.

All smart contracts are stored in their own blockchain to increase efficiency. Nodes in the system use the public keys of the nodes to verify the contract before it is stored in the blockchain.

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 - graph to show procedure

5.2.2 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. The purpose of the blockchain is that two different parties cannot claim the same unit of produced energy, or the transaction payment.

5.2.3 Settlement

When the system receives information of consumed and produced electricity of a node, the smart contract is self-enforcing in the manner that these events trigger functions that control the settlement.

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.

Each node in the system has a bill of how much the node owes other nodes in the system, and how much other nodes owe this node. The bills are accumulative and assumed settled at regular intervals, e.g. each month. It is assumed that all

nodes with transaction info has one, and only one contract, at any given time. Furthermore, it is assumed that any excess electricity from a seller is stored in battery, and a buyer will get it's electricity from pure producers if a seller can not deliver the sufficient amount at a given time.

5.2.4 Consensus

The consensus protocol is modeled after the Raft protocol. Five nodes are in charge of the consensus process in the consortium blockchains. The reason why only five nodes should participate in the consensus process is to reduce the message overhead in the system. The consensus process has a randomly selected leader. Any node can be part of the consensus process, and any node can become leader. If a node participating in the consensus goes offline for some reason, another node will take its place. Nodes in the system 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.

5.2.5 Website

The website is the user interface to the system. 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. The log in is done with the user's public and private keys. 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

6.1.1 Python

Python [59] 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. One of these libraries is the unit test framework, which supports test automation and eases code testing.

6.1.2 Twisted

Twisted [60] 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 new connections and lost connections of peers.
- **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

6.2 Blockchain Modules

This section contains descriptions of the modules implemented in the blockchain layer.

6.2.1 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 [61] 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. The *hexdigest* method is used to return a string object of double size, containing only hexadecimal digits, for better readability.

Other methods in the class include *validate_block* for validation of a new block, based on the previous block: *propose_block* to create a new block, based on the

previous block and the current time; and *assert_equal* to verify that two blocks are identical.

6.2.2 Network Module

A peer-to-peer(p2p) network is implemented in the network module, using the Twisted [60] 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 was 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 received are processed in the factory, and further handled by the *Node* object.

6.2.3 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 making 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.

6.2.4 Consensus Module

The consensus module used in this system is Raft.

The consensus module 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 vote in that term or not 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.

If the *append entries* RPC includes a new proposed block, the follower will validate the block and write the block to the log during the next run of the *state machine*.

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, and is triggered when the parties of the contract have produced and/or consumed electricity.

Three assumptions were made for the settlement process in the system.

- During any interval, the amount of electricity consumed in the system equals the amount of electricity produced in the system.
- 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.

The settlement process is accumulative and based on previous data generated in the system. Each node has a *account* where settlement info is stored.

6.3.2 Smart Contracts

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.

6.3.3 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 prosier availability.

Chapter 7

Testing

This chapter describes how the system was tested. Section 7.1 describes the method for how the complete system was tested. Section 7.2 specifies how 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:

- 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 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 module 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 Website From Consumer Perspective

Monitoring tested using simulation data Smart contract outlined Only tested locally

7.2.2 Settlement

While the functionality of the settlement module can be verified through the system test, a unittest was also conducted to ensure correct behavior.

7.2.3 Unittest of Blocks module

The block module was also tested through a unittest.

7.2.4 Consensus

Leader election time for different number of nodes Maximum number of nodes participating in consensus tested.

7.2.5 Network Module

Several aspects of the Network module were tested:

- Message latency from point to point.
- Latency of detection of new connections and closed connections.
- Communication with two machines on the same network.
- Communication with two machines on two separate networks.

Chapter 8

Result

Success for two machines on same network

Fail for two machines on different network, due to port forwarding issues

Show result of wallets from two nodes with smart contract, include extract of testdata in testing?

8.1 System Test

Verify settlement with extract of simulation data and settlement result Size of blockchain - size per block

Chapter 9

Discussion

Correctness Strong Consistency Efficiency - update log is very inefficient Store all peers since finite size of network? No protection against brute force attacks on network IP/port

Logical p2p, cannot prove or guarantee that the electrons from one producer ends up at the consumer of which it has a smart contract with. Consumers pay the producer for the electricity they have consumed and the producer produced, even if it is not the exact same electricity.

advantages of building own blockchain and smart contract system is that it ensure zero transactional fees in the system. more control over data? disadvantages include maintenance of code and documentation, more work trough implementation and testing

settlement and storage in parallel. use transaction blockchain to audit that settlements done right, immutable properties ensures correct information

What differs in this solution, compared to the other solutions from the background chapter? DESCRIBE SHORTCOMINGS IN SYSTEM IN RELATION TO DESIGN - Communication: open connection between all nodes, limit to e.g. 5 and broadcast numbered messages until all nodes have received them. - Consensus: limit consensus nodes to only 5.

Reasons for RAFT

Discuss how the functional specs are satisfied or not, refer to by number

Space saving - delete smart contracts that are no longer valid.

RAFT is not PBFT. Anyone can establish themselves as leader and start sending out *AppendEntries* messages. In the presence of Byzantine nodes in the system, this is not suitable. By making new users part of the validation process, the risk

is reduced. Since this system is implemented in a restricted physical area, where participants are neighbors and possibly acquaintances, the risk of Byzantine nodes is reduced even further.

9.1 Privacy

9.2 Future Work

Current implementation stores blockchain and other logs locally on machine, a future improvement could be to store the data in the cloud to save space.

To save space, blocks containing transactions that are settled or smart contracts that are no longer valid can be deleted.

Make new users part of validation - to ensure true consortium. Current solution can be attacked through brute force, since the only requirement to join is to know the IP address and port number of an existing node on the network.

Automatic selection of who to initiate smart contract with for consumer based on pre-defined constraints

Only some nodes part of validation? and let these have open connections to each other, let remaining nodes have connections to at least one of the validation nodes.

Merkel trees for scalability. Is there a point though, since merkel trees are for time saving (?) and we are not iterating through blockchain to find previous transactions. Maybe usable in smart contract blockchain

Increase BFT by signing messages to prove authenticity.

Automatic smart contract based on pre-set prices given by prosumer and consumer

Settlement - better algorithm - including taking electricity from batteries for better flow

Chapter 10

Conclusion

To build such a system on top of existing blockchains such as Bitcoin or Ethereum, may result in a costly system, due to relatively large transactions fees and many, low-value transactions in the microgrid network. Ethereum, in addition, requires Ether to run applications/smart contracts on the EVM. Using the modular blockchain of Hyperledger might be a better idea, however - some cost?

The blockchain implemented in this thesis is able to run and process all transactions in the network at no additional cost.

Consortium - node ids are linked to identities - if a node tries to attack the blockchain, its identity will be known and actions may be taken thereafter.

Transparency - account owners can at any time review there account actions. Super user can see everything

Problem - all users can review all other users account activity through the blockchain ledger - however it is a rigorous task to read all account activity from the ledger. Thus, the application layer exist to make it easier to monitor account activity - and this is only available to the account owner.

Appendices

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.g0_LP24.

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

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

[12] 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.

[13] *Virtual Money - at Your Own Risk*. University of Luxembourg. Nov. 2014. URL: https://wwwen.uni.lu/university/news/latest_news/virtual_money_at_your_own_risk.

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

[20] 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.

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

[22] 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.

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

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

[25] 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.

[26] 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.

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

[28] 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.

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

[30] *Hyperledger Fabric Ordering Service*. URL: https://github.com/hyperledger/fabric/blob/13447bf5ead693f07285ce63a1903c5d0d25f096/orderer/README.md.

[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. *Hyperledger Fabric*.

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

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

[46] 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.

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

[48] 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/.

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

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

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] Coinstar. *NAD Grid - Thomas Sun - APAC Blockchain Conference 2018*. Youtube. Mar. 2018. URL: https://www.youtube.com/watch?v=euTqN3ZJE00.

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

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

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

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

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

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