Norwegian University of Science and Technology

Department of Engineering Cybernetics

Use of blockchain technology for settlement in a finite energy market (microgrid)

By:
Marit Schei Tundal
marittu@stud.ntnu.no

Supervisor: **Geir Mathisen**

April, 2018

Abstract

Sammendrag

Abbreviations

RPC	=	Remote Procedure Call
EVM	=	Ethereum Virtual Machine

Contents

Chapter 1

Introduction

Provides a trustworthy foundation in many applications. Saw need for this after the housing bubble burst in 2009 and the Lehman Brothers fell. There is no longer the need for a trusted institution to handle transactions - due to the blockchain technology, trust is no longer need at all to be part of transactions of value.

Distributed - not sending copy of assets. Prevents double spending of assets like money, votes, electricity etc.

This thesis mainly consists of two parts: implementation of a blockchain for storing data in a distributed network; and a outer API for settlement of energy transactions in a microgird. The focus lies on the implementation of the blockchain.(?)

accounting system - just like money issued by banks and government

Chapter 2

Background and Related Work

This chapter contains background theory on blockchains and microgrids. Related work on the subject is presented at the end of the chapter

2.1 Microgrids

Microgrids or smartgrids

2.2 Blockchain

A blockchain is a decentralized database, 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 [1]. In the following sections of this chapter, some key features about blockchains, the technology behind blockchains and, their usage will be analyzed and discussed.

2.2.1 The Blockchain Basics

The blockchain is primarely 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 as a whole, and the mathematical functions behind it, and not the individual participants [2]. Due to the decentralization of the network, there is no single-point-of-failure, as the database is duplicated in every node of the network. If a new user enters the network, or if they have been offline for a while, they can obtain the most recent version of the blockchain by querying the network multiple times, until they are convinced they have the correct version.

The below description of a blockchain is based on the Bitcoin blockchain. Many of these features 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. An input is a previously received and unspent output, containing the amount of an asset. 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, specifying the amount in each, 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 messaged, by using public and private keys. Each node can obtain a random private key. Based on this, a public key is generated from an Elliptic Curve Digital Signature Algorithm (ECDSA) 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) 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; the merkel root of the merkel tree containing all the transactions in this block; and a nonce and a target used to calculate the hash of the new block. The merkel tree and how the hash of a block is found, will be discussed later in the chapter.

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 exsists 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 differend 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 [1]. 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 alterations 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.

2.2.2 Private, Public and Consortium Blockchains

Blockchains are usually divided into three different categories, depending on the way they are governed. [3] 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 does not have a single, central authority controlling it. Another, and wider classification is permissioned and permissionless blockchains. These are categorized depending on whether the blockchain is open to anyone or if some form of permission is required for access to the blockchain. Public blockchains are categorized as permissionless, while private and consortium blockchains fall under the category of permissioned. The three main categories will be further discussed below.

Public Blockchains

A public blockchain is completely unregulated. It is open to anyone that wishes to read it; transactions are completely transparent. It is also open to anyone wishing to make transactions or participate in the consensus process for validating blocks - making it fully decentralized [3]. The only requirement is a valid transaction.

Public blockchains rely heavily on cryptoeconomics for security in an otherwise low-trust environment. Cryptoeconomics is described as "*a combination of*

economic incentives and cryptographic verification using mechanisms such as proof of work or proof of stake, following a general principle that the degree to which someone can have an influence in the consensus process is proportional to the quantity of economic resources that they can bring to bear" [3].

The most important innovation from public blockchains, is the problem of who transmits first and the issue of double spending Additional benefits of the openness of a blockchain includes the resistance to censorship. A single node or a group of nodes, provided they do not control a big enough (depending on the consensus model) portion of the network, cannot keep transactions out of the blockchain.

Public blockchains can grow quite large, and posses huge amounts of processing power, combined. This computational capacity can in turn be used for ...

As mentioned, public blockchains rely on cryptoeconomics which in turn requires computationally heavy consensus models. Proof of Work (PoW) is the most common protocol used to reach consensus in a public blockchain. PoW has its drawbacks, including vast amounts of computing power required, especially in the Bitcoin blockchain

Private Blockchains

Trusted parties Lost identification credentials - private keys Reverse verified transactions in event of e.g. system error, fraudulent activity, mistake in recipient address. Reduced access lowers risk of outsider attack Benefits of blockchain require some for of mistrust between users, otherwise a shared database might be a better solution Developers might change system without the users consensus Key issuing from e.g. government to be allowed participation - loses part of the aspect of being independent from third parties. Ability to revert transactions, e.g. property ownership in legal dispute Trust comes from being trusted to participate in verification No single point of failure, as not dependent on one single computer Cheaper due to no transaction fees Little or no chance of 51% attacks

Consortium Blockchains

Summary

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

2.3 Consensus Models

Consensus models are important in distributed systems, in order for ... correct version of the shared ledger. Safety, liveliness, fault-tolerance Two types of fault-tolerance - fail-stop and Byzantine faults

2.3.1 Proof of Work

Computationally expensive Requires very much energy Variable called nonce, which is adjusted in order to find a hash matching requirements of another variable called target Consensus used in Bitcoin untrustworthy peers wanting to modify past blocks have to work harder than peers wanting to add new blocks. Target value determines difficulty, the lower the target value the more attempts are needed on average to find ha hash beneath the target threshold. Mining to generate more crypto currency not needed or desirable in this use case Brute force Good scalability for participating nodes and completely decentralized Slow, energy waste, latency

Hardware centralization - ASCI

2.3.2 Proof of Stake

Tries to solve the energy problem in proof of work Rich get richer More assets in network - more likely to be chosen as validator for mining - reward Centralized

Casper Tendermint no coin creation (mining) exists in proof of stake. Instead, all the coins exist from day one, and validators (also called stakeholders, because they hold a stake in the system) are paid strictly in transaction fees.

2.3.3 Raft

Raft is a partially asynchronous, leader based consensus algorithm for managing replicated logs across the system.

There are three main parts of the Raft algorithm: leader election; log replication; and safety.

Properties that are true at all times:

- 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

A node can be in one of three states at any time. The possible states are leader, follower, or candidate. When the system operates under normal conditions, there is exactly one server and all other nodes are followers. During the election of a new leader, there can be one or more candidates. The followers are passive, and only respond to requests from leaders or candidates.

Terms

Remote procedure calls (RPC)

In Bitcoin, transactions are never finalized, in the sense that they can become part of a fork. As time passes and more blocks are created, a transactions 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 assumes trusted members, thus no Byzantine faulty nodes.

Ripple requires each node to have a list of trusted nodes, each nodes list must have a 40% overlap with other nodes in the network. Consensus in multiple rounds

2.3.4 Practical Byzantine Fault Tolerance

2.3.5 Summery of Consensus Models

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, there does not exist a single best consensus model. Limitations and restrictions are present in regards to power consumption, scaling, security and transaction speed. These factors will be further discussed in the following section.

2.4 Security and Weaknesses in Blockchain technology

Centralized the mining to countries with cheap electricity - vulnerable to changes in policy on electricity subsidies

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 Ethereum. Some of these blockchains will be discussed in this section.

2.5.1 Bitcoin

How Bitcoin works, was roughly described in section 2.2.1. 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 [4], 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 [5].

Another major advantages of Bitcoin is transferring money abroad [6]. Compared to traditional transfers between banks which are both costly (on average 7.45% [7]) 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 [8].

An example of an application built on top of the Bitcoin blockchain is Lighthouse [9]. Lighthouse is a crowdfunding application where users can make donations in Bitcoin to projects.

2.5.2 Ethereum and Smart Contracts

Ethereum is an open-source blockchain application platform [10] proposed by co-founder Vialik Buterin [11] in 2013 in his white paper [12]. 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 [13], 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 programmable blockchain. Protocols run on the Ethereum Virtual Machin (EVM) [13]. 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, the ether has a purpose other than simply being a crypto currency. In order to run applications on the Ethereum blockchain, a small fee, or gasprice, is required. This is to ensure efficient and economical code, as each computational step in the code requires a certain amount of gas [12].

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 has been created on top of the Ethereum blockchain. Among these is the Brooklyn Microgrid, which will be further discussed in section 2.6.1. Other applications include Decentralized News Network [14], where factual news reports are rewarded with tokens and free of censorship; uport [15], 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 [16]. 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.*" [16]

The Fabric blockchain is a base for developing blockchain with modular architecture [17], 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 Proor-of-Concept (PoC) testing [18]. 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 [19]
- eVoting [20]
- Peer-to-peer energy trading [21]
- Cross-border trade operations [22]

2.5.4 Comparison of Commercial Blockchains

Table 2.1 summarizes the key features of the previously discussed blockchains:

Table 2.1: Comparison between blockchains [1, 13, 16]

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 Related work

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

[23] : Current trading of renewable energy without blockchain – pay ahead. NRGcoins directly correlating to energy transferred to grid – pay for use Calculations for buying and selling energy. First to propose this kind of system?

2.6.1 Brooklyn Microgrid

Virtual/physical grid Based on ethereum, but not scalable so making its own

2.7 Other Use Cases

Land ownership Sharing economy Transferring money (abroad) Intellectual property rights - music, kodak etc. Crowd funding via smart contracts

Chapter 3

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.

3.1 Application From User Perspective

- 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.
- The user can buy tokens that are later used as payment for consumed electricity. The price of a token is directly related to NOK.
- Tokens can be exchanged back to NOK at any time.

- A website lets users set up smart contracts and monitor production and consumption of electricity.
- The website requires a log in where users are identified by their node id.
- Prosumers with surplus electricity to sell, put up their availability on the website.
- Consumers who want to purchase electricity can query available producers and initiate a smart contract.
- When the contract is in place and energy is transferred, the production and consumption can be monitored on the website.

3.1.1 Use Case

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 an 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 receives tokens from the consumer for the consumed electricity. The tokens can be converted into NOK at any time, or be used to purchase electricity from other prosumers on the grid. 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.

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, meaning that the user cannot purchase electricity from other prosumers during the duration of the contract. However, the user can choose not to consume electricity from the grid by e.g. using less energy or getting electricity from another source, such as a battery

or a generator. The user must purchase tokens which are used as payment for consumed electricity of the grid. Consumers can at any time log in to the website and monitor how much electricity they consume at minute intervals, and at what price.

3.2 Application From Computer Perspective

- The computer is connected to a smart meter and receives information about a node's consumption and production of electricity at periodic intervals.
- The machine passes the readings on to the blockchain node.
- When two users initiate a smart contract, the machine passes it on to the blockchain for validation and storage.
- New readings from the smart meter are processed on the website and added to the graphs for consumption and production.

3.2.1 Use Case

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 further processing. The reading is also passed to the monitoring display.

Monitoring Consumption and Production

When new measurements are received from the smart meter, the computer updates the graph on the website to include the new measurements. 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.

3.3 Blockchain

- The blockchain receives two types of input from the application, a smart contract between two users and electricity transactions.
- The blockchain is distributed between all the nodes in the network.
- The nodes broadcast their individual transactions across the network.
- The leader of the blockchain proposes a new block at minute intervals, containing all new electricity transactions from that period of time.
- The proposed block is broadcasted to all network nodes, who validate the block and the transactions.
- If a majority of the nodes deem the block valid, it is stored in the blockchain.

3.3.1 Use Case

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 4

Implementation

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

- Python for implementation of the system
- Python framework Twisted for network communication
- Website?

4.1 Python

Python [24] 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 and eases code testing.

4.2 Twisted

Twisted [25] 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 utilize 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.

Chapter 5

Design

5.1 System Architecture

FIGURES

5.1.1 Blockchain Layer

5.1.2 Application Interface Layer

5.1.3 Interaction Between Layers

5.2 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 [26] 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.

5.3 Network Module

A peer-to-peer(p2p) network is implemented in the network module, using the Twisted [25] 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 clients 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 to send 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.

5.4 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.

5.5 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 canceled 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*.

5.6 Storage Module

In the *storage* module, tasks related to reading and writing to file are handled.

Each node has four different logs, all containing information that must be kept intact in the case of a system failure.

- **Config log**: Contains the node id, public and private key pair
- **Vote log**: Id of node who received vote for each term

- **Proposed blocks log**: Each log entry consists of the index, term and the block proposed by the leader
- **Blockchain**: Blocks in the actual blockchain

All new entries are appended to the top of the log, for more efficient reads. The *proposed blocks log* also contains a method for deleting entries in case of conflicts. The *blockchain log* is final and consistent across all nodes.

5.7 Message Module

The message module consists of the different messages that are sent between the peers in the system. Messages are sent between Twisted connections using transports and the built in Twisted method *sendString*.

5.8 Smart Contracts

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

Chapter 6

Testing

Test with at least 10 nodes Ideally test over Ethernet/several computers

Chapter 7

Result

Chapter 8

Discussion

Possible to add one more message type - ack propose block and commit several blocks to the blockchain at a time and at longer intervals 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

8.1 Conclusion

8.2 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. (Crashes, but not so relevant since data is duplicated across the network)

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

Appendices

Bibliography

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

[2] 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.

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

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

[5] 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/.

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

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

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

[9] 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/.
[10] *Ethereum - Blockchain App Platform.* URL: https://www.ethereum.org/.
[11] Alyssa Hertig. "Who created Ethereum". In: *Conidesk* (Mar. 2017). URL: https://www.coindesk.com/information/who-created-ethereum/.
[12] Vitalik Buterin. *Ethereum White Paper.* Tech. rep. Ethereum Foundation, 2013. URL: https://github.com/ethereum/wiki/wiki/White-Paper.
[13] *What is Ethereum?* URL: http://www.ethdocs.org/en/latest/introduction/what-is-ethereum.html.
[14] *News by the people, for the people.* URL: https://dnn.media/.
[15] *Open Identity System for the Decentralized Web.* URL: https://uport.me.
[16] Linux Foundation. *About Hyperledger.* URL: https://www.hyperledger.org/about.
[17] Linux Foundation. *Hyperledger Fabric.*
[18] Linux Foundation. *Industries.* URL: https://www.hyperledger.org/industries.
[19] *SWIFT's Blockchain Pilot For Bank-To-Bank Transfers Went Extremely Well.*
[20] 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.
[21] *A peer-to-peer platform for efficient trade of electricity.* URL: https://www.nadgrid.com/.
[22] 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/.
[23] 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.
[24] *Python Software Foundation.* URL: https://www.python.org/.
[25] *Twisted Matrix Labs.* URL: https://twistedmatrix.com/trac/.
[26] *hashlib - Secure hashes and message digests.* URL: https://docs.python.org/3/library/hashlib.html.