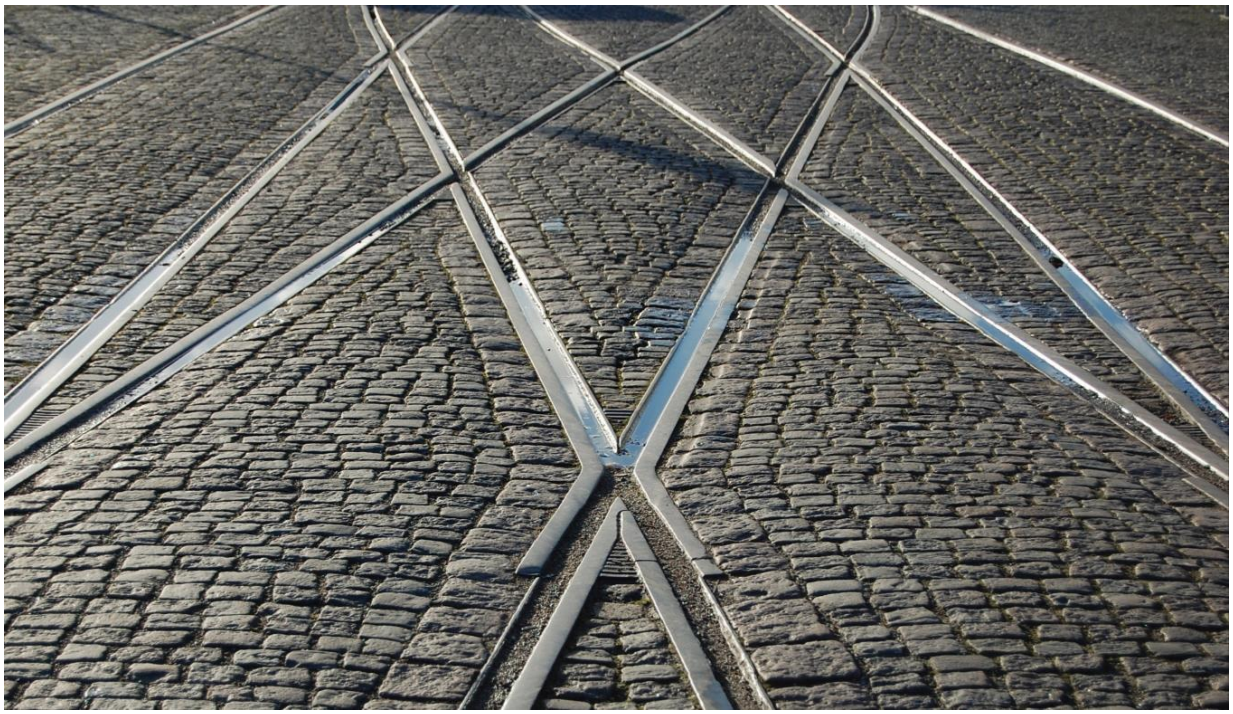
CHALMERS



Toward a framework for assessing meaningful differences between blockchain platforms

Bachelors Thesis in Industrial Engineering and Management

JESPER ALM
JOHN LINDBLAD
JONAS MEDDEB
PHILIP NORD
KRISTOFFER SÖDERBERG
JAKOB WALL

Department of Technology Management and Economics
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sweden 2019

Toward a framework for assessing meaningful differences between blockchain platforms

JESPER ALM
JOHN LINDBLAD
JONAS MEDDEB
PHILIP NORD
KRISTOFFER SÖDERBERG
JAKOB WALL

Supervisor: Peter Altmann
Examiner: Erik Bohlin
Department of Technology Management and Economics
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone + 46 (0)31-772 1000

Cover:
Photo by Natalia Y on Unsplash

Department of Technology Management and Economics
Gothenburg, Sweden 2010

Abstract

Blockchain technology has gained public interest as the technology behind cryptocurrencies. However, there is a potential for the technology to be applied in additional contexts. One way to implement a blockchain solution is through use of a blockchain platform. There are today many different platforms to choose from. Unfortunately, blockchain platforms differ in the way that they operate and function. There is a lack of frameworks for stakeholders to compare different blockchain platforms. The purpose of this study is to provide a framework for comparing different blockchain platforms. Further, the purpose is also to use the framework to identify meaningful differences when developing a ROSCA application.

This study was conducted by an initial literature review of blockchain technology and blockchain platforms to identify interesting properties. Three platforms, Ethereum, Hyperledger Fabric, and Neo, were thoroughly examined. A ROSCA application was developed on Ethereum, identified platform properties were evaluated, and finally meaningful properties for the ROSCA application development were determined.

Thirteen platform properties were found and used as the structure for the platform evaluation. Information about the three platforms was then structured to enable a juxtaposition with each of them. The development of a ROSCA application showed that important properties for a ROSCA application are trust, privacy and access, handling payments, cost of development, and developer community.

This study form a foundation for further research about how to compare blockchain platforms. The selection of blockchain platform properties can be used as a framework to either evaluate blockchain platforms or in a comparison to other future frameworks. The development of a ROSCA application showed meaningful differences for development of that certain type of application. However, to get a more nuanced view on meaningful properties for different applications, further research, including development of other applications, as well as on other platforms, is crucial.

Keywords: blockchain, blockchain technology, blockchain platform, Ethereum, Hyperledger Fabric, Neo, ROSCA, framework, decentralized application

Sammanfattning

Blockkedjetekniken har fått ökat allmänintresse genom sin roll som den grundläggande tekniken bakom kryptovalutor. Det finns potential för att tekniken också kan skapa nytta i andra kontexter. Ett sätt att implementera en blockkedjelösning är att använda en blockkedjeplattform och idag finns många olika plattformar att välja mellan. Dessa skiljer sig dock i sätten de fungerar på och det saknas sätt att jämföra olika blockkedjeplattformar. Syftet med denna studie är att skapa ett ramverk för jämförelse mellan olika blockkedjeplattformar. Vidare är syftet att använda ramverket för att identifiera meningsfulla skillnader vid utvecklingen av en ROSCA-applikation.

Denna studie genomfördes genom en inledande litteraturstudie av blockkedjeteknik och blockkedjeplattformar för att identifiera intressanta egenskaper för en jämförelse. Tre plattformar, Ethereum, Hyperledger Fabric och Neo undersöktes grundligt. Senare utvecklades en ROSCA-applikation på Ethereum och de identifierade egenskaperna i hos plattformarna utvärderades, vilket slutligen resulterade i en redogörelse för de meningsfulla egenskaperna för ROSCA-applikationen.

Tretton plattformsegenskaper hittades och användes som strukturen för plattformsutvärderingen. Information om de tre plattformarna strukturerades sedan för att möjliggöra en tabellstruktur vid jämförelse. Utvecklingen av en ROSCA-applikation visade att viktiga egenskaper för en ROSCA-applikation är tillit, integritet och tillgång, betalningshantering, utvecklingskostnader och community för utvecklare.

Denna studie utgör en grund för ytterligare forskning om hur jämförelse av blockkedjeplattformar bör genomföras. Valet av meningsfulla egenskaper hos blockkedjeplattformar kan användas som ett ramverk för att antingen utvärdera blockkedjeplattformar, eller för jämförelse med framtida ramverk. Utvecklingen av en ROSCA-applikation hjälpte till med att identifiera meningsfulla skillnader för en ROSCA-applikation. För att få en mer nyanserad och generell kartläggning av meningsfulla egenskaper för olika applikationer krävs ytterligare utveckling och forskning, både av ytterligare applikationer och på andra plattformar.

Contents

Chapter 1

Introduction

The public interest in *blockchain technology* has grown rapidly in recent years. As the key component for cryptocurrencies, the attention for blockchain technology increased as the interest in cryptocurrency reached a peak at the beginning of 2018. Cryptocurrencies then had a total market cap of above 800 billion USD, with *Bitcoin* as the flagship (CoinMarketCap, 2019). Besides acting as the foundation for cryptocurrencies, blockchain technology was assessed as one of the hottest emerging technologies in *Gartner's hype cycle*, both 2017 (Panetta, 2017), and 2018 (Panetta, 2018).

Interestingly, demonstrations of blockchain's supposed benefits remain scant and there is significant confusion around what a blockchain actually is and how it can and should be used. For instance, definitions of blockchain technology diverge. Some highlight its immutability, others its decentralization. This divergence causes confusion and possible misunderstanding of what blockchain technology is and how it differs from, and relates to, existing technologies.

The lack of a common definition can be illustrated by the following three blockchain definitions:

> *"A system in which a record of transactions made in bitcoin or another cryptocurrency are maintained across several computers that are linked in a peer-to-peer network."* (Oxford Dictionary, 2010)
>
> *"Blockchain is a shared, immutable ledger that facilitates the process of recording transactions and tracking assets in a business network. An asset can be tangible or intangible."* (IBM, 2019)
>
> *"The blockchain is a decentralized ledger of all transactions across a peer-to-peer network."* (PwC, 2016)

The three above definitions have fundamental differences. The Oxford Dictionary declares blockchain to be solely used to record *cryptocurrency* transactions while IBM is slightly more general; describing blockchain technology to record transactions and tracking assets in *a business network*. PwC is being less specific by suggesting that the ledger simply holds *all transactions*. Other discrepancies of blockchain definitions exists, not only between but also within industries (Jeffries, 2018).

Furthermore, it is not only the definition of a blockchain that might vary, but also the blockchain implementation. Design parameters of a blockchain make the technology modular and allow it to be optimized for different situations, resulting in a vast variety in the way in which a blockchain might function. All of this contributes to challenges when developers seek to leverage this emerging technology to realize their business ideas.

Specifically, developers may build applications on various *Blockchain platforms*. These platforms are established networks or frameworks for developers to use in order to apply blockchain technology for their specific purpose. Many developers and other kinds of stakeholders are examining the possibilities to develop blockchain applications. Thereby, they face one main challenge: there exists no framework for comparing platforms to build the applications on. Without such a comparing framework, developers risk selecting a suboptimal blockchain platform for hosting their application.

The issues with lacking a framework for comparison have been examined in previous research. One example is a systematic review of current research on blockchain technology that emphasizes the need for objective evaluation criteria for comparing different blockchain solutions (Yli-Huumo, Ko, Choi, Park, & Smolander, 2016). The versatility that characterizes blockchain platforms and the modularity of the technology might be problematic when selecting a suitable platform for a specific application. One possible way to facilitate this process would be to define a framework that articulates differences between blockchain platforms to help developers analyze the platform's potential for their application.

1.1 Problem statement

There are many blockchain platforms providing blockchain solutions to developers among different industries. However, there is a lack of frameworks for comparing blockchain platforms. This problem has been identified and discussed for private blockchain platforms (Tuan Anh Dinh et al., 2017) but not yet as a comprehensive assessment for both private and public blockchain platforms.

Through a clearly articulated assessment framework, describing meaningful properties and differences, developers are better able to select the appropriate blockchain platform for a specific purpose. This makes it easier for anyone wanting to develop some blockchain application to choose a platform that is a good fit for the application in terms of matching the required attributes for the application with the properties of the platform. An assessment framework also contributes to the common understanding of blockchain technology by highlighting different key aspects of blockchain platforms that either retain or evade certain system behavior.

1.2 Purpose

The purpose of this study is to provide application developers with a framework to compare different blockchain platforms, by mapping differences between platforms, and identifying which of them are meaningful through developing an application.

Chapter 2

Background

This chapter gives an introduction to theoretical concepts required to follow the work presented in the rest of the report. It begins with an explanation of blockchain technology, focusing on how blockchain data structures can be used to store data in an append-only ledger secured by cryptography. Followingly, distributed applications and existing issues regarding decentralization of such applications are presented. Different ways of how blockchain technology can help manage such issues, while at the same time upholding the integrity of the ledger, are then covered. An introduction is thereafter given to blockchain networks as application platforms, and how these can enable decentralized applications by hosting services defined using smart contracts. The chapter ends with a description of the ROSCA system which was implemented in the study, as an example of an decentralized application.

2.1 Blockchain Technology

Since there is no clear definition of what a blockchain is, this section focuses at a selection of technologies present in blockchain platforms. These technologies are relevant for understanding a blockchain as a type of distributed ledger, in which obtaining consensus of the state is a decentralized process.

2.1.1 Ledger and Data

Traditionally, a ledger is a physical book used for keeping records, such as those related to financial transactions or civil registrations. This fits the books' physical nature since records in ink can be appended easily, but are difficult to modify without leaving traces. For a data structure, this is a property called append-only, meaning data can be written, but neither deleted nor modified. A data structure with this property can be used as a ledger, and updating the ledger state can then be done through recording data transactions. A data transaction takes a valid state and modifies it in some way, rendering a new valid state. When keeping record of all transactions, the current state of the ledger can be calculated by processing these transactions in order. In blockchain systems, such state model is referred to as unspent transaction output (UTXO) (Narayanan et al., 2016). This, since the balance of an individual equals the unspent transaction outputs assignable to that individual when processing all conducted transactions. Cryptocurrency payments are value transactions between different parties. In a ledger, these value transactions can be recorded using data transactions modifying the state. This

type of data transaction deducts a given amount of currency from a sender, and adds the same amount to a receiver (Narayanan et al., 2016).

When making digital ledgers, several issues arise in the lack of limitations of the digital format. One of these is how digital files can be easily manipulated. A hash-linked list is structured to work around this issue. The structure is based on linking together ledgers through the use of cryptographic hashfunctions. These mathematical functions can render a hash, which is a unique replicable fingerprint, of some kind of input data. This allows a ledger to reference all transactions in another ledger by appending only the hash of that ledger, as seen in Figure 2.1. By allowing ledgers to be broken down into blocks, which use hash references to refer to previous blocks, a chain is created, in which blocks can not be modified without modifying the hashes of the following blocks (Narayanan et al., 2016). Since all data in the previous blocks affect the final hash, integrity is kept through the aggravation of data manipulation.

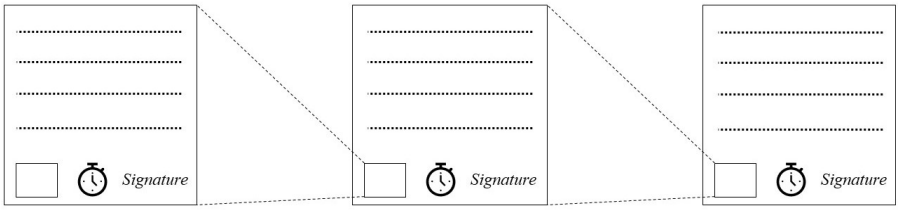


Figure 2.1: Linked ledgers through hashing (Interpretation of figure from (Narayanan et al., 2016))

As the state of the ledger can be calculated by processing its transactions, storing the state explicitly is redundant. However, as the ledger grows, the amount of transactions required to process to access the ledger state increases. Different variations of the data structure use more or less redundancy to find a balance between utility and size. As the shape of a data structure has an impact on the time complexity for searching and modifying the data, the network can be optimized by developing and selecting efficient data structures. One way to do this is by implementing a tree structure for the ledger, such as a Merkle tree seen in Figure 2.2. While this is only one data structure, different blockchain implementations may use different data structures, making different trade-offs suited for their application. The Ethereum Modified Patricia Merkle Trie (Wood, 2014) is another example of how these data structures are modified and tailored for specific usages.

2.1.2 Distribution

An important difference between physical and digital ledgers is the way digital ledgers can be systematically distributed. The way distribution of data between computers in a network is performed has evolved together with the growth of the internet. Distributed applications consist of at least two communicating parties. In many applications, these parties can be modelled as a client and a server (van Steen & Tanenbaum, 2018). The

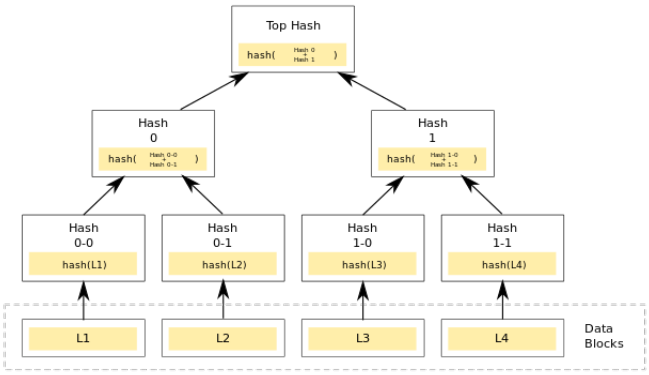


Figure 2.2: A Merkle tree (David Göthberg, 2019)

client is the software which runs on the user computer, commonly referred to as an application distributed to the user. The client provides the user interface of the application, which lets the user interact and communicate with the server. The server is the software which runs ”behind the scenes”, on a computer governed by the application provider. It performs computations and responds to requests on data as a service. The server also holds or communicates with the database, where application data is kept.

2.1.3 Decentralization

Distributed applications let users interact with each other through the use of a third party. This has opened up for new types of applications which have disrupted traditional industries (Zervas, Proserpio, & Byers, 2017). However, relying on a third party to serve the application has some limitations. Decentralization removes the need of a trusted third party, and has potential to enable transparent, trustless transactions of data and value directly between parties (Wright & De Filippi, 2015).

Peer-to-peer networks are the first step towards decentralization, as they let peers connect to a network of other peers directly. In these logical networks, peers may distribute data between each other using their respective node in the network. The computer node does not need to know every other node in the network, since nodes communicate using a common network protocol to efficiently offer the service to its peers. This protocol defines what rules nodes need to enforce in order to behave like a trusted node (Narayanan et al., 2016).

Since a node in a network may be malicious, this protocol needs to handle byzantine fault. Byzantine fault is defined in the byzantine generals problem as traitors able to send incorrect information to the loyal generals, who need to agree on the same plan (Lamport, Shostak, & Pease, 2002). If the loyal generals are able to decide upon the

same plan of action, while not knowing which of the other peers are traitors, their tactics are byzantine fault tolerant. In a network where certain nodes are trusted, this problem can be solved using digital signatures. It becomes more difficult however, when the intent of the other generals is unknown. This allegory, explaining the problem of trustless coordination, has greatly influenced research on consensus in distributed systems (Lamport et al., 2002).

When different versions of the ledger state is created, either by time differences or malicious nodes, a fork occurs. By implementing a byzantine fault tolerant consensus protocol, the network assures that there is consensus about the state of the distributed ledger. This, in a decentralized network, in which the integrity of the ledger state is upheld without the need of a central authority. No such protocol is however tolerant to hard forks, i.e. when some peers implement a different version of the consensus protocol, incompatible with the previous one. Such scenario renders the ledgers incompatible and peers neglecting the other ledger as invalid, resulting in two separate networks (Narayanan et al., 2016).

2.1.4 Consensus mechanism

In a decentralized network sharing a distributed ledger, the agreement on a the ledger state is called consensus. Reaching consensus is a main challenge of decentralization (Narayanan et al., 2016). In order to reach consensus, the network protocol incorporates a consensus mechanism consisting of a consensus model and a consensus algorithm (Altmann, 2019). These can be implemented in various ways making the network behave differently. When these two are combined in a sensible way the blockchain network can be seen as byzantine fault tolerant.

Consensus model

The consensus model describes how the nodes allowed to create a new block on the blockchain are selected (Altmann, 2019). The two selection procedures encountered within this report are selection through randomization and selection through delegation. A randomized selection process requires that the node that gets to create a new block is selected in a randomized way. A selection through delegation on the other hand is conducted by peers on the network electing other nodes, in which they trust, to be block producers and to maintain the integrity of the network.

Consensus algorithm

The consensus algorithm describes how the validation of a new block is performed (Altmann, 2019). There is a number of different consensus algorithms implemented on different blockchain networks. These can be described as competitive based and elective based (Altmann, 2019). The choice of algorithm impacts the finality of validation. Probabilistic finality on the other hand means that the transaction included in a

block on the chain can be reverted and that the risk of that transaction being reverted is probabilistic.. The other case immediate finality does not allow a validated block to become invalid (Vukolić, 2015).

Proof of work (PoW) is a commonly used competitive based algorithm for reaching consensus on blockchain networks. The fundamental concept is that nodes can participate in the consensus process by performing computational puzzles to which there are no quick solution. When a node completes a puzzle, it gets to create the next block on the chain. Nodes validate the block by adding its hash in the next block extending it. Since the chain is implemented as a linked list, only one chain can be the valid one unless a fork has been made. This results in a racing game for the nodes to solve the puzzle and thus being able to add a new block to the chain. Since computational power is used for solving the puzzles, the protocol states that the chain with most work done on it, meaning the longest one, is the valid one and that any other chain should be discarded (Narayanan et al., 2016). A prominent drawback of the PoW algotithm is that it is slow and can be very energy consuming (Wang, Zheng, Xie, Dai, & Chen, 2018). The benefit is that it is the only objectively verifiable method known to reach consensus (Altmann, 2019). Objectively verifiable means in this context that a node, with no other knowledge except what is defined in the protocol, and that follows the consensus algorithm, will come to the exact same conclusion as the rest of the network regarding the current state. The opposite of objectively verifiable is subjectively verifiable. Subjectively verifiable means in this context that different nodes in the network come to different conclusions regarding the state when following the consensus algorithm. This requires a large amount of social information, such as reputation or identity, to participate in the consensus process (Vitalik Buterin, 2014).

Another competitive based consensus algorithm is proof of stake (PoS). Proof of stake is based on the concept of acquiring a stake in the system to participate in the validation of blocks. If a node agrees that a block is valid, it can validate this by placing a bet, or stake, on that block. A benefit of this algorithm is that it is energy efficient. A drawback is that it lacks objectiveness in its consensus procedure. (Wang et al., 2018).

As opposed to competitive based algorithms, the elective based algorithms require some type of already established trust in the party elected to propose and validate blocks. The benefit of elective based algorithms is that they offer higher performance than competitive algorithms, with the drawback that they require a pre-established trust in the validating party (Altmann, 2019).

Practical byzantine fault tolerance (pBFT) is an algorithm developed in 1999. Consensus in a network implementing pBFT is maintained by a certain set of nodes, where one is the leader node and the other are used as backup. When a request is sent on such network, the leader node informs the backup nodes about the request, and then executes the request. The leader node then informs the sender of the request result, who in turn awaits the backup node's response to come up with the same request result. The integrity of such system is maintained through a majority vote that the request result is correct. In order to guarantee integrity of the data in such a system it requires $3f + 1$ nodes to be able to tolerate f faults for the network to stay fault tolerant (Castro & Liskov, 1999).

The delegated Byzantine fault tolerance (dBFT) is a consensus algorithm that operates in the same way as pBFT but with a variation. In this set up nodes are part of consensus through a system of proxy voting where the nodes, can vote for the validator they wish to support. When the elected validators reach consensus around the transactions to be included in a certain block, the block is generated and finality is achieved (Wang et al., 2018).

2.2 Blockchain Platforms

Decentralized applications do not conform to the traditional client-server paradigm since there is no central server. Instead, a decentralized application hosts its service on a decentralized peer-to-peer network, letting clients interact with the application through nodes on the network. Blockchain platforms support the development of new types of decentralized applications (Buterin, 2019). This, through providing required resources and infrastructure for execution, networking and data management. First in this section, the application software that developers write for blockchain platforms, commonly referred to as smart contracts, will be explained. Subsequently, parameters that vary throughout different platforms' designs will be presented. The design choices give varying system performance, why the aspects of performance relevant for this study will afterwards be covered. Ending the section is an explanation of some of the resources helpful for developers on platforms.

2.2.1 Smart contracts

Application specific softwares executed by blockchain platforms are often referred to as smart contracts. Because this software is distributed on the blockchain network, consensus can be obtained on the code and how it should be executed. This uplifts the software from being just code, to being a policy or a contract that peers can execute, to enforce certain behaviour in how the state of the ledger should be modified. This type of contract can represent the rules between two business parties. When a customer and the owner of a vending machine interacts, the transaction is also automated. The cost that the customer has to pay for a specific candy bar, and whether change is payed out, are rules which are enforced by the machine. While a vending machine can act incorrectly and it may be difficult for the customer to be justified for execution fault, smart contracts are a way to make transactions predictable and transparent (Narayanan et al., 2016).

2.2.2 System design choices

To understand the varying range of blockchain implementations existing, system design choices, and their implications from different perspectives, need to be described. The perspectives include which the participants are in the network, how they can interact

with the system, and what amount of privacy and access that is granted to different participants. Further, important aspects regarding non-technical components, such as incentive models and cost structures, are also presented in this section.

Participants

Every peer connected to a peer-to-peer network is running a node on the network. However, a peer can behave in different ways, depending on in what way it intends to interact with the network, and which functionality the network enables. Thus, the explanation of what a node is needs to be broadened to fully understand how a peer can connect and interact with the network.

A node is called a full node if it stores the entire blockchain (Satoshi Nakamoto, 2008). If a full node creates and validates blocks on the network it can also be referred to as a validating node (*NEO Documentation*, 2019). If a peer runs a full node on a network implementing a PoW consensus algorithm, and creates blocks, the peer is also called a miner. (Satoshi Nakamoto, 2008).

A peer who performs transactions on the network can be seen as an owner of an address from which transactions can be sent and received (Satoshi Nakamoto, 2008). These transactions can consist of cryptocurrencies, but also of some arbitrary data (Buterin, 2019). Owning an address and making transactions often require the use of a full node, but it does not require that the address owner runs this full node by themselves. In order for an owner of some address to make transactions, without the need to store the entire blockchain, some blockchains implement a protocol called simplified payment verification (SPV). Although, if they do not run a full node, they cannot participate in the consensus mechanism (Satoshi Nakamoto, 2008). Since the structure and implementation varies among different blockchains, some blockchain networks enables the use of what is called a light node. A light node is a node that only requires the storage of a small part of the blockchain to verify transactions. Neither light nodes can participate in the proof of work consensus algorithm (Buterin, 2019).

To ease the management of a user's address a wallet is typically used. A wallet is a software that keeps track of a peers cryptocurrency, manages all the details of their public and private keys, and makes things convenient by providing the user with an interface through which transactions can be made (Narayanan et al., 2016).

Interacting with the system

As already described in 2.1.1, all interaction with a decentralized network is based on data transactions. In the context of blockchain platforms, such are commonly referred to as simply transactions. A study of how participants can interact with a blockchain network includes how transactions are conducted and what types of transactions there are, as well as what different system operations a participant can get to execute through transacting with the network.

Making transactions on a blockchain network requires computation. To compensate the system, doing transactions may require a fee. This fee can be of different quantity depending on what is specified in the protocol, what type of transaction it is, and how much computational power it consumes. The transaction fee is in systems using PoW given to the owner of the full node that created the block in which the transaction was included (Buterin, 2019).

Two concepts regarding performing transactions related to blockchain networks are on-chain and off-chain transactions. On-chain transactions constitute the transactions made directly on the chain by broadcasting them to the network, whereupon full nodes record them and add them to the blockchain. Off-chain transactions are instead enabled through private communication. They are not broadcasted immediately to the blockchain, and relies completely on external recording to eventually be bookkeeped on the chain (Gudgeon, Moreno-Sanchez, Roos, McCorry, & Gervais, 2019). Off-chain transactions do not need to wait for all the nodes on the network to come to consensus before getting the transaction verified, resulting in off-chain transactions being faster than on-chain transactions. It can be rational to perform off-chain transactions as appose to on chain transactions as long as the transaction fees are lower than the ones offered on-chain (Gudgeon et al., 2019).

Transactions are as mentioned the instrument for getting to execute operations on a blockchain network. The operations available on blockchain platforms are often similar to those present in traditional computing or database systems. Some networks, such as Ethereum, constitutes their own virtual machines, i.e. systems serving similarly as ordinary computers (Buterin, 2019). Examples of operations available in such virtual machines are writing to storage, reading from storage, and doing simple computations such as additions (Wood, 2014).

Access and privacy

Important issues are how anonymous a user is on a network, and how accessible to others a user's transactions are. This, because the participants interacting on a blockchain network need to understand who can gain access to and read their data. Different degrees of privacy and access can be suitable for different purposes.

Peers connecting to a blockchain network can have a varying degree of anonymity when identifying themselves. The anonymity in part reflects the degree of privacy and depends on how the network is designed. When peers identify themselves on the network, it is usually done through the use of digital signatures (Narayanan et al., 2016). How the actual identification is performed can vary on different blockchain networks. For instance, on some networks the user is only known through their public key which represents the address from where they make transactions (Narayanan et al., 2016)(Wood, 2014). Other networks enable the functionality of the peer choosing whether to be anonymous or not (*NEO Documentation*, 2019).

The provided anonymity made possible by digital signatures is secure, but can be compromised if the user is careless. On some networks a user's real identity could for

instance be compromised if a user often makes use of the same public key when making transactions with cryptocurrency. The reason for this is that the network will store all transactions ever made by that address and by using the same public key as identifier the transactions could be linked to ones real identity by, for instance, analyzing ones purchase patterns (Narayanan et al., 2016).

Linking to the introduction's stated confusion concerning concepts in blockchain technology, some networks use the term private blockchain and public blockchain while others use the term permissioned and permissionless blockchains. This has to do with whether the access to the blockchain can be restricted or not. If a blockchain is public or permissionless it is accessible by anyone, while a private or permissioned blockchain is invitation based (Hyperledger, 2018). As will be shown in the platform section, some blockchains implements a hybrid solution to the two.

User access

The hardware required to interact with a blockchain network varies depending on how a peer intends to interact. If a user intends to run a full node, it only needs a computer that has internet connection and disk storage to hold the entire blockchain (Narayanan et al., 2016). If the intent is to run a light node less disk space is needed. If a peer intends to perform mining, in theory, nothing more than what was stated for a full node is needed. However, in practice, a miner will constantly need to upgrade its hardware for better computational performance, since the computational puzzles constantly increases in difficulty (Narayanan et al., 2016).

Incentive models

Since the blockchain network is upheld by nodes run by peers on the network, it would stop working if nodes suddenly stopped participating in the consensus mechanism. The way of giving peers an incentive to maintain this procedure is referred to as incentive models. Understanding the incentive model on a blockchain network is important to make a decision on whether to put in the effort to act as a validating node, as well as whether to trust that the existing validating nodes are acting honestly or not.

When a miner finds a block, and this block is accepted onto the blockchain, in networks implementing PoW, the miner of that block is rewarded with some value unit, commonly a network specific cryptocurrency. (Wood, 2014)(Satoshi Nakamoto, 2008). Finding a new block is a probabilistic task and since it may take a long time before this happens, the phenomenon of a mining pool has come in to place. A mining pool is a group of miners forming a pool, in which all are attempting to find the next block, and all have a designated recipient of the mining reward. This designated recipient is called the pool manager, who will receive the entire reward when a block is found. The pool manager then distributes the reward for finding the block depending on how much work every miner has put into solving the puzzle. The pool manager can also take a cut for himself for managing the mining pool. The use of a mining pool increases the

probability for a miner to get paid and the larger the mining pool the more attractive it becomes, since it then gets more probable that the next created block will come from this mining pool (Narayanan et al., 2016). This has resulted in the fact that most active miners are connected to a mining pool, and that there are only a few different mining pools doing the total mining performed on the largest proof of work blockchains. This is an important aspect to understand when considering security aspects and networks' degree of centralization (Etherscan, 2019).

The reward system in PoW networks are present in many networks with other consensus mechanisms such as PoS and dBFT. Similarly, validating nodes are rewarded for adding blocks to the chain, with an on chain transaction of some value unit. However, in private blockchains, such as Hyperledger Fabric, transaction fees are not necessary. This means validating nodes must be compensated either with off chain value transactions or by gaining trust or reputation on the network (Hyperledger, 2018).

Hosting costs

In order for a miner to draw a conclusion on whether it is logical to act as a validating node, or for a developer to decide to launch an application on the specific blockchain, it is imperative to understand the underlying cost structure of that blockchain network. The cost structure consists of everything that the user will have to pay for in order to operate on the network and will, as shown, depend on what interaction the peer intends to make. Whether the transactions are made on-chain or off-chain can have a strong impact on the total expenditure of a users transactions. When considering a network's cost structure the overhead costs may also needs to be taken into account. For instance, if a peer operates a validating node on a proof of work system then the overhead costs would include buying mining equipment and paying for energy consumption (Satoshi Nakamoto, 2008).

2.2.3 System performance

Aspects that describe the performance of a blockchain network have in this study been divided into computing performance and system stability. Computing performance is about a networks capacity to process transactions and operations, whereas system stability is focused on the risk factors and reliability of the system.

Computing performance

In the aspect of computing performance, this study has mainly focused on transaction throughput and how finality is achieved. Transaction throughput is measured in transactions per seconds (TPS), which is the amount of transactions managed to be performed on the blockchain per second. The limitation of the networks' throughput has to do with the fact that each block is hard coded to be of a certain size and therefore it can only register a certain amount of transactions. Additionally, since transactions are

stored in blocks being created and added to the chain by nodes, the networks throughput is also determined by the networks block time. Determining the networks' TPS is done by calculating the amount of transactions stored in a block divided by the block creation time (Narayanan et al., 2016). The network throughput is essential in order to understand how many concurrent transactions a system can handle. In this study we will refer to a networks transaction speed as the inverse of the transaction throughput.

A closely linked subject to transaction throughput is how finality is achieved. The two types explained in this study are immediate finality and probabilistic finality. The pBFT is an example of a consensus algorithm that offers immediate finality while the PoW algorithm is an example of an algorithm that offers probabilistic finality (Vukolić, 2015). In networks providing probabilistic finality, users are usually recommended to wait for a couple of block extensions before the transactions in a block are considered finalized. The reason for this is that the probability of reverting the process diminishes exponentially as more blocks extend the block holding the transactions to be considered finalized (Satoshi Nakamoto, 2008). Since network latency increases with the number of nodes in the network, the time to reach finality in a network is affected by both network latency and the implemented algorithm for consensus.

In blockchain networks the block time can vary. As previously stated, the PoW algorithm is the only objectively verifiable algorithm known, which in turn can offer the network a high degree of decentralization if implemented in a sensible way. The downside, as stated, is that it is slow in comparison to a, for instance, pBFT algorithm. The reason that it's slow is partly because it sets a difficulty in how long on average it should take for a miner to find a new block. There are blockchains that uses 10 minutes as block time and as we will see, others that have an average block time of around 10 seconds (Buterin, 2019). From the above calculation of transaction throughput it can be concluded that the transaction throughput will increase with shorter block time, but there are downsides with implementing a short block time, which will be discussed in the following section.

System stability

System stability is in this study the aspect of how susceptible a blockchain is to system disruption and its degree of centralization. A problem with implementing a PoW algorithm that uses short block times is the increased probability of stale blocks and forks. A stale block is a block that is found by a miner, deemed valid under protocol but not accepted on the chain due to the fact that another miner also found a block and it was that block that got accepted onto the long term chain. Since only one block is included in the long term chain the other block and all resources allocated into mining that block goes to waste. Shorter block times therefore increase waste, can deter miners from mining, and thereby stop them from upholding the security of the system (Buterin, 2019). Furthermore, latency in the network will make different nodes hear about the broadcasted blocks at different times. The nodes will follow the protocol, which in PoW blockchains means always extending the block that they heard about

first. This results in some nodes extending one of the blocks and other nodes extending another, thereby increasing the risk of creating a fork on the chain. Additionally, short block times in PoW algorithms also adds to the security concerns. One of those security concerns is that larger mining pools will have an increased probability of getting their blocks accepted. This, while smaller mining pools more frequently will see their mined blocks orphaned, simply due to their small size. This can negatively impact the networks degree of centralization. (Buterin, 2019).

Another issue that arise with competitive based consensus algorithms is the issue of a 51 percent attack. This is if an adversary controlling a large amount of the validating nodes on the network was trying to compromise the system. There is nothing magical about the 51 percent number, it only states that if someone controls a majority of the validating nodes, then they could potentially compromise the system by subverting consensus to their favour. This is possible since this single actor would then most probably be the one to find and create most of the new blocks.

A possible way to compromise a POW system with a 51 percent attack is through a double spend attempt. A double spend attempt is conducted by performing a cryptocurrency transaction to someone and then revert that process by spending the same cryptocurrency on a fork of the blockchain and make that the long term valid one by adding blocks in a more frequent pace than the rest of the network. In theory, an adversary could manage such attack with even less than majority control (Narayanan et al., 2016). It has been shown that only 25 percent of the computational power can be sufficient to forge blocks (Bach, Mihaljevic, & Zagar, 2018).

When a blockchain implements an elective based algorithm, it meets other security issues. One of these are that, since an elective based consensus algorithm puts trust in certain validating nodes, the system could be compromised by any of the validating nodes if they would have such intentions. A system implementing an elective based algorithm could be considered to have a high degree of centralization, since the network is governed by a small amount of nodes (Altmann, 2019).

As blockchain networks evolve changes to the software may need to be made. When the software gets updated there could be a situation where some nodes have updated the new software but some have not. This would mean that some nodes would extend blocks that other nodes would consider invalid and thereby ignore. This could result in the nodes working on two different chains that both sides will consider to be the longest and valid one. This causes a hard fork and the risk of this happening needs to be considered when implementing changes to a blockchain network. (Narayanan et al., 2016)

The above section represents a few different ways blockchain networks can be disrupted. Another aspect when considering system disruption is the amount of nodes that constitutes the network. Since it is proven that byzantine generals problem fails when one third of the nodes are malicious or faulty and that the competitive based algorithms are sensitive to 51 percent attacks proven to be possible with 25 percent network control, the number of validating nodes on the network is an important aspect to consider when estimating the networks risk of disruption.

2.2.4 Application development

Beyond the implementation of a certain blockchain with its properties, the viability of an application platform also depends on how accessible the solution is to the development team. The accessibility of an application platform benefits on how open the platform is, as well as the size of the developer community, contributing to the available tools, documentation, and development. This ties into how platforms which strive towards open technology lowers development costs of applications because it encourages contribution from the developer community (Holzer & Ondrus, 2011).

Open technology is also a key part of decentralized applications claiming to be trustless. This requires the network protocol and the code of the application to be open source. The trust of a blockchain platform can be built on the basis of transparency, relying on experts to review the systems for laymen to trust. Another way open source allows the quality of the application to grow is when different parties can govern the smart contract and client implementations. This opens up for several different clients to compete of who gets to deliver a given service to the user (Vrba, 2018).

2.3 Zamfir's triangle

There are models intended to better visualize certain aspects of different blockchain platforms. Zamfir's triangle is one, used in this study to justify choosing the three different that were examined. Vlad Zamfir is one of Ethereum's lead researchers and thus exceedingly involved with blockchain platform development. Zamfir has proposed a model, illustrated in Figure 2.3, for describing trade-offs in fault tolerant consensus mechanisms for different blockchain platforms. This model visualizes that consensus mechanisms are unable to achieve low latency finality, low overhead cost, and have many validating nodes at the same time (Zamfir, 2017).

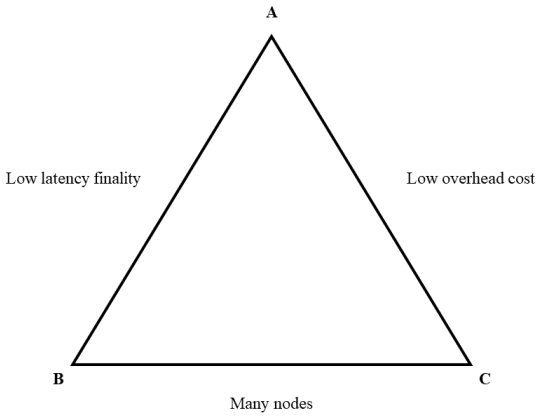


Figure 2.3: Zamfir's triangle illustrating the fundamental trade-offs in fault tolerant consensus mechanisms. (Interpretation of Zamfir's original tweet (Zamfir, 2017))

The triangle is created so that corner A, as referenced in Figure 2.3, includes platforms with both low latency finality and low overhead, but with few nodes. Corner B includes platforms with low latency finality and many nodes, but high overhead. Finally, corner C includes platforms with low overhead and many nodes, but with high finality. The consensus mechanism of different platforms can, according to Zamfir, be described as a specific point inside or on the edge of the triangle (Zamfir, 2017). However, it should be noted that this model only describes the consensus aspect of a platform.

2.4 Rotating savings and credit association

This section describes the application developed for the study. Initially, an introduction to rotating savings and credit associations (ROSCAs) as systems is presented, followed by the pronouncement of its different design components. All information is intended to facilitate the development described in chapter 5, and to allow for later assessment and discussion.

2.4.1 Introduction to ROSCAs

ROSCAs are informal financial institutions for peer-to-peer saving and ledning, found primarily in developing countries (J. Besley, Coate, & Loury, 1993). The motives for ROSCA participation, described by Rangarirai Mbizi and Edson Gwangwava, are, among others, the need to acquire consumer durables, insurance, and intrahousehold

conflict in resource allocation (Mbizi & Gwangwava, 2013). ROSCAs were historically formed with periodical contributions of grain of different sorts. Today, however, ROSCAs generally use money as the subject to be transferred between its members (Kabuya, 2015).

There are different ways to implement a ROSCA, but the basic idea can be described by a general example (Ambec & Treich, 2007). Interested parties come together as members of a group. This group holds regular meetings where every member contributes to a common pot. At the end of a meeting, the common pot is given to one member of the group. This member is then excluded from receiving the pot at future meetings, but continues to contribute to the pot. When every member has been assigned the pot once, a cycle is finished and the ROSCA either continues for another cycle or is disbanded.

2.4.2 Rotation

The rotation of a ROSCA refers to the way in which the participants receive the pot. For the system to be fair to its participants, the pot rotates between the members so that every member has received the pot once every cycle. It is possible to create associations without the rotation component where some participants obtain the pot every time while others solely fund the pot. This type of association is no longer a ROSCA but instead called an accumulating savings and credit associations, ASCA.

There are two distinct types of rotation implementations called random and bidding (Kedir, 2015). A random ROSCA selects the order in which the members receive the pot, at random. The member receiving the pot is still excluded from getting the pot at future meetings, and thus only get the pot once every cycle. The difference for a bidding ROSCA is that the members competitively bid for the pot at every meeting. The pot is allocated to the highest bidder, which then is excluded from future biddings.

There are different ways to implement the bidding procedure. One possible bidding technique, and the one used for this study, is the Vickrey auction. This is a type of sealed-bid auction where participants submit bids without knowing the bid made by other bidders. The highest bidder wins the auction but only pays the second highest price. Note that the highest bidder in this auction is the one who can spare the biggest amount of money. This could be implemented in a ROSCA by simply having the auction winner get the pot for the second highest price every meeting, and be dispossessed of the right to place future bids. This procedure thus obtain a rotation for the receiver of the pot.

2.4.3 Transactions of value

Transactions of value need to be handled within the association, and more specifically between the members and the current receiver of the pot. This could be implemented in a few different ways, the simplest being to physically exchange money during an

actual meeting. However, more modern ways of handling ROSCA transactions are either by digital direct payments, or by using a digital common pot or fund possessed by a member or a third party.

For certain implementations of the rotation, a common pot or fund is vital. This is the case when implementing a rotation component as a Vickrey auction. As a consequence of this, there needs to be a possessor of the pot or fund and thus be trusted by all members of the association. This gives motive for implementing the ROSCA as a distributed and decentralized application, given that the pot could be distributively possessed.

2.4.4 Joining the circle

In order for a ROSCA to be established and operate it needs to have onboard members. Member groups of traditional ROSCA implementations often include friends and family. As these groups have a natural connection, they obtain the vital trust within the association. However, as one major benefit of ROSCAs is the distribution of risk within the group, it could be argued that ROSCAs with member diversity is to prefer.

One way to establish trust within a group is to write a common contract. Blockchain platforms enable writing contracts online and keeping them decentralized. This can theoretically allow members to be anonymous and yet participate in the association. Furthermore, trust can be increased by specifying acceptance requirements for the ROSCA participation.

2.4.5 Penalty

Another important aspect of a ROSCA is how to handle unwanted behavior among members of the association. One example of such behavior is not paying the contribution to the common pot. The association needs to decide on appropriate penalties for such event, giving participants incentive to act honestly. This aspect of a ROSCA ends up outside the scope of this study. It is although necessary to mention the penalty as a component in order to communicate an thorough design.

Chapter 3

Method

This chapter describes different aspects of the method used for this study. The first section specifies the research design and aims to facilitate understanding and analysis of the method used to obtain the results presented in chapters 4 and 5. This is followed by a declaration of the limitations connected to the research design. Finally, an evaluation of the method is provided.

3.1 Research design

The study aimed to obtain a gradually improving platform comparison framework, based on different kinds of empirical data. Data was gathered by going through technical documentation on platforms, and by documenting the development process of a ROSCA application. The framework for platform comparison was continuously updated and revised. This was, however, all following the first challenge, to, through a literature study, understand and describe blockchain technology, despite the conceptual confusion discussed in the introduction.

3.1.1 Initial literature review

The study was initialized with a literature review on blockchain technology. By consulting an expert in the area, the authors' of this study were recommended the study literature "Bitcoin and Cryptocurrency Technologies" by Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller and Steven Goldfeder. This literature was published by Princeton university press in 2016 and provides the reader with an explanation on how the well known Bitcoin network is implemented, along with its underlying blockchain technology. Topics covered are the ones presented in the background chapter.

Other sources has continuously been added to supplement the understanding of the various fields included in blockchain technology. This information has come from a variety of places, ranging from twitter posts from leading developers on a specific blockchain platform to scientific articles found by Google scholar and Chalmers University library database. Specific search words included "proof-of-work", "off-chain transactions", or "distributed ledger technology". It was however found that scientific articles describing blockchain technology on a generally applicable level was scarce.

Reading the abstract of an article was usually sufficient to assess its relevance and weather to save it for the study or not. Additional sources found through different Google searches could include newspaper articles and blog posts. To ensure the quality of the blog posts, the focus was on posts written by people with a high reputation in the blockchain community. Examples could be developers working for organizations such as Ethereum. When using documents from web pages we downloaded the information to assert that the information would be recoverable in the case that the web pages changes or goes down. Finally, during the course of this study we have also used unpublished information from ongoing current research provided by an expert on the area working for RISE - Research Institute of Sweden.

A principle that has permeated the writing process for this report is an avoidance of giving assertive definitions of confusing concepts. For such concepts and terms, the definition presented is the one used in this report, and not necessarily the one that is considered the correct definition by everyone.

3.1.2 Platform selection and examination

Three platforms were selected and examined. The selection was made by mapping platforms to Zamfir's triangle and selecting platforms near its different corners, to ensure a platform selection with as much variety as possible. Hyperledger Fabric, Neo, and Ethereum were the three selected platforms.

The platform examination was conducted through reading technical documentation, provided by the platforms' original creators. The examination of Hyperledger Fabric is based, unless otherwise stated, on the official documentation found on the Hyperledger web page (Hyperledger, 2018) (Hyperledger, 2019b). The examination of Neo is based, unless otherwise stated, on the Neo organization's technical documentation (*NEO Documentation*, 2019). The examination of Ethereum is based, unless otherwise stated, on the Ethereum whitepaper (Buterin, 2019) and yellowpaper (Wood, 2014).

The structure of the platforms' description, found in Chapter 4, was set during the initial literature review of blockchain technology and blockchain platforms. The structure details can be reviewed in Appendix A. The order of the information about the platforms was coordinated based on the structure of the platforms' documentation, and with the intention to maximize readability. The structure can also be found in Appendix A. Once examined and structured, information about the platforms was synthesized and structured in a conforming way to enable a juxtaposition with each of them.

3.1.3 Development of ROSCA application

The synthesis of platform information, structured in different categories, were used as a draft of a comparison framework, for mapping meaningful differences between the examined platforms. The draft was first used to select one designated platform

to develop an application on. The choice fell on Ethereum, of reasons which will be described in chapter 5. The development consisted of designing and implementing a ROSCA. ROSCAs appeared among other alternatives during idea brain storming, and was eventually picked primarily based on the reasonable scope of developing it and its compatibility with being implemented on a blockchain.

At the beginning of the process, information needed to develop on the platform was collected by using the documentation provided by platform creators and development community. The reason for this was to put together a course of action related to constructing the application. After that, the development of the application could take place, ending with a functioning ROSCA implemented and deployed on a blockchain network. The development continued as long as the time limit allowed.

Each member involved in the development process continuously documented their work in a diary. The diary contained information related to what had been done, which obstacles were encountered, and how they were solved. At the end of the development process, the diaries of each participant were analyzed. The aim was both to distinguish which properties were of most importance when developing the ROSCA. However, a reflection of whether the properties are of importance for other applications as well was also initialized.

3.1.4 Concluding establishment of the final framework

Ultimately, the framework was reconsidered through discussion with input consisting of data, both from the external examination of the platforms and from the experience of developing a ROSCA application on Ethereum. Firstly, all differences between platforms, collected during the development process, were discussed and evaluated. Secondly, certain differences were pointed out as more or less meaningful when developing a ROSCA application. Even if the development only was carried out on one platform, hypothetical differences between platforms were discussed with the use of the data gathered in the platform examinations.

3.2 Limitations

The duration of the study was no more than about four months, and consequently, only three platforms were examined. Other platforms could have been studied to provide more details for the comparison framework. Regarding the development, there are two distinct limitations. The first is that the development of a ROSCA was only conducted on one platform due to the time limits. The second is that the development only resulted in a minimal viable product and thus only highlights situations and obstacles occurring in the initial phases of development. With more time, the development would also include testing the application live with real participants.

3.3 Method evaluation

This evaluation starts off with a critical review of the various sources used for data gathering. As scientific literature on blockchain is scarce, a critical approach to sources is crucial to ensure the quality of the study. Afterward, a discussion regarding the platform selection is held, unearthing the implications of the specific choice. Lastly, some concepts regarding the quality of the study are discussed.

3.3.1 Source evaluation

The concept of source evaluation that has been used during the study is called functional source concept (Blomkvist & Hallin, 2015). The concept is best described by the quote, "no source in itself is bad or worthless" (Blomkvist & Hallin, 2015). This implies that it is more important how to use sources rather than what sources to use. However, it is important to note that sources that are not academic have not gone through the process of peer review. This needs to be taken into account when assessing their credibility. Examples of such sources being used are articles from web pages, blog posts, and the ongoing work of unpublished material provided by an expert in the area.

Social media, blogs, and forums have increased in importance since information is more and more available online (Shneiderman, 2016). It implies that academia no longer is at the forefront of research in all areas. The consequence of this is that crowdsourcing is becoming an increasingly powerful way of conducting research. This becomes highly relevant for this study since blockchain has become closely associated with decentralization through its peer-to-peer characteristic. Therefore, the importance of combining applied and basic research have increased (Shneiderman, 2016). Applied research is characterized by a mission-driven and a solution driven approach, in which the aim is to find practical guidelines. The ideal scientific approach is therefore deselected in favor of a more realistic approach to finding the answers wanted. Basic research is more traditional, theory-driven, and uses classical scientific principles. The combination of these two is what is recommended, and what this study builds on.

3.3.2 Platform selection

The choice of platforms had to be carefully considered to ensure diversity among the three platforms that were enough to provide a solid comparison framework for later assessment. The three platforms were selected with the use of Zamfir's triangle and by using the fact that these were close to the triangle's extremes. The extremes refer to the corners A, B, and C in Figure 2.3. The reason for choosing platforms mapping to the extremes in Zamfir's triangle is that this was believed to be the most suited way to select three platforms that potentially could generate the greatest variety for

discussing meaningful differences. The three platforms are not important themselves but are chosen to represent a diversity to form a framework from.

3.3.3 Contribution of the development

The development was necessary as an input to the discussion about meaningful properties. However, the character of the development as a tool for research comes with some limitations. Firstly, the method is subjective in the meaning that it is reliant on the knowledge and efficiency of the developers. It is possible that another development team would have encountered other obstacles while developing the same application. Since only one specific application was implemented, on a particular platform, there are limitations to the insights' general applicability, which is further discussed in the 6.

3.3.4 Generalizability and usefulness

A qualitative research study could be evaluated by using a couple of quality criteria (Flick, 2014). Those extra debatable in the context of this study will be discussed, namely generalizability and usefulness. Generalizability can be described as the degree to which the result of the study can be generalized outside the context of this study. Usefulness measures whether the conclusions from this report can be used by potential readers in their everyday lives (Flick, 2014). Generalizability and usefulness are closely intertwined for this study. Even if the three platforms studied are three of the most used among blockchain platforms, the conclusions are much more useful if the resulting framework for comparison can be used when choosing between other platforms too. The most persuasive argument for the generalizability of this study is that the platforms were selected to be as different as possible according to Zamfir's triangle. The platform examination questions presented in Appendix A were used when comparing the three platforms in this study. These questions were developed over time, and new questions were added when new differences were identified. This structure of questions could be generalized and used for other platforms too even if some tweaking of it may be required. This means that the quality of the output from this study if this proves to be true and in terms of generalizability and usefulness, is to be considered strong

Chapter 4

Platforms

Three different blockchain platforms were chosen because of their difference according to Zamfir's triangle (Zamfir, 2017), illustrated in figure 4.1. The triangle symbols a three-way tradeoff between latency, overhead cost and number of nodes. First, Hyperledger Fabric is a framework for designing networks with low latency finality, low overhead and a few nodes. Second, Neo consists of an own network, also with low latency finality, but with high overhead and many nodes. Third, Ethereum is a network that has high latency, low overhead and many nodes. The platforms have been examined and later evaluated for their suitability for developing the ROSCA system described in section 2.4. In this section, platforms will first be described individually, to afterwards be juxtaposed with each other in a synthesis table in section 4.4.

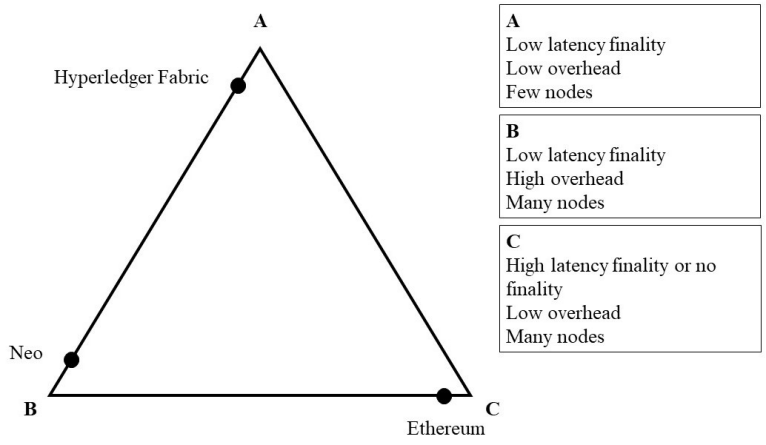


Figure 4.1: Zamfir's triangle with Hyperledger Fabric, Neo and Ethereum (Interpretation of original figure from Vlad Zamfir's twitter (Zamfir, 2017))

4.1 Hyperledger Fabric

Hyperledger Fabric is one of six open source blockchain solution frameworks made available by Hyperledger, hosted by the Linux Foundation. Hyperledger provides both blockchain frameworks and several tools for additional functions. The basic idea for Hyperledger Fabric is allowing different companies and organizations, not necessarily

trusting each other, to share a global up-to-date state of data that are in all parties' interest.

A possible alternative solution would have been for each company to supply their own data storage. The problem would then be that only the company itself could change the data. If the state of the data demands other companies' and organizations' input in order to be consistent, this solution is not effective. To keep a consistency of data, a third party, trusted by all involved parties, could keep the data and distribute it. This requires a third party that everyone trusts and that can provide the demanded services. Thus, this solution also establishes an undesirable, low fault tolerance.

The solution Hyperledger Fabric suggests is for the involved companies and organizations to share a distributed ledger. The companies will each hold an access node necessary in order to reach the information and modify the current state of the data. The parties state a way in which the permitted distributed system should operate and can thus all hold and manipulate the ledger while always being consistent with other network participants.

The Hyperledger Fabric framework is modular and allows permitted users and creators to modify the network configuration in order to make the network suit a specific purpose. A network can consist of many channels, and each channel is associated with its ledger. A channel is a group of organizations that share a distributed ledger, and thus can share private information with each other.

4.1.1 System design

A Hyperledger Fabric network is built around one or many distributed ledgers. The blockchains are permissioned which, in practice, means that only permitted organizations have access to the blockchain data. All network participants need to be connected to an organization node in order to get access to the blockchain.

The configuration of a Hyperledger Fabric network is as already stated modular which means different parts of it can be adjusted freely to suit the creator's purposes. In the network configuration, it is stated which organizations have access to what parts of the network. The network configuration is established by the initial network creator which could be either an organization within the network or a third party.

Data structures

As illustrated in figure 4.2, the ledger of Hyperledger Fabric is comprised of two data structures. One is representing the ledger state, and one is constituting the blockchain. The ledger state is implemented using a database, holding a cache of the current ledger state. The blockchain does not store the data itself, but rather the history of transactions leading up to the current state.

The database keeping the ledger state is exchangeable. The state database could be a graph store, a relational database, or a temporal database. Thus, the structure of

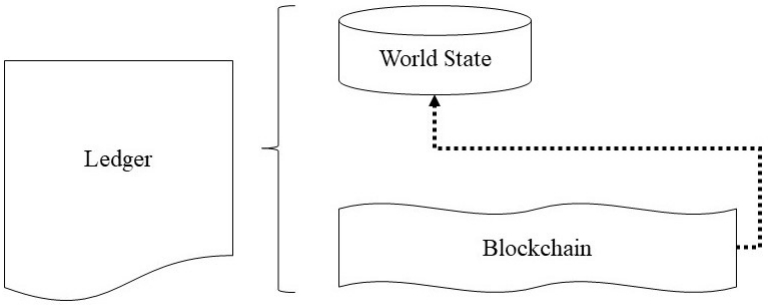


Figure 4.2: Modelling of Hyperledger Fabric's ledger (Interpretation of figure from Hyperledger Fabric documentation (Hyperledger, 2019a))

the data may vary between networks. Regardless of the implementation, the database is updated for every new block of transactions appended to the ledger, ensuring the database contains the current ledger state. LevelDB is the default implementation of the database which uses a key-value structure to store its data. For ledgers that are structured as JSON documents, CouchDB is described as the preferred database to use. It structures its data as JSON objects, which enable nesting of data points into objects and arrays.

The blockchain is structured as sequentially linked files, illustrated in figure 4.3. Each file is referred to as a block and contains three sections. The block header contains the block number, the hash of the previous block, and the current block hash. A list of transactions is gathered as the block data. Finally, the block also contains block metadata containing information about certificates, the time when the block was created, and the signature of the block writer.

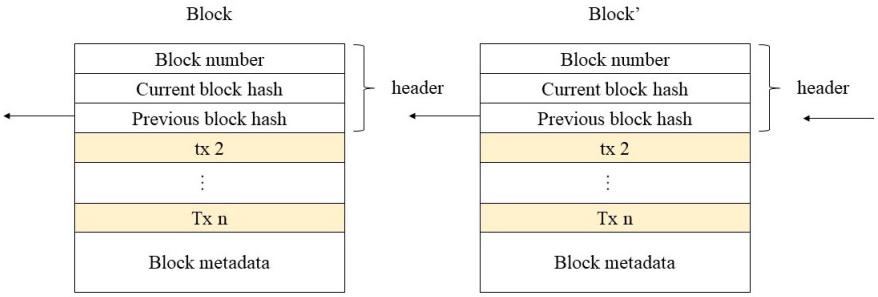


Figure 4.3: Model of Hyperledger Fabric's blockchain (Interpretation of figure from Hyperledger Fabric documentation (Hyperledger, 2019a))

Participants

There are different actors with different roles and capabilities within the Hyperledger Fabric network. Two different network participants play a vital part in the flow of the network.

The first of the two is a peer node. To access a ledger, an organization need to run a peer node. A peer node is a container in which the owner organization's different channel ledgers are situated. The container also stores and executes the channel chaincode, which is Hyperledger's counterpart of a smart contract. The node is unique to every network organization. If an organization require lots of interactions with the network, it is possible to possess many peer nodes connected to the same channel.

The other participant is the ordering node. The best way to describe an ordering node is as the administrative point of the network. The ordering node is provided by an organization and initiates the network configuration and thus the channels. A network can contain many ordering nodes from different organizations. In that case, the ordering nodes together shape the administrative service for the network.

Consensus mechanism

As described in 2.1.4, the consensus mechanism can be divided into two parts: a consensus model, and a consensus algorithm. The consensus model defines the network of participants engaging in the consensus process as well as important properties and rules. The ordering nodes, as the administrative service of the network, are defined in the network configuration. This configuration also defines the endorsement policy of the network.

The consensus algorithm is a three-step process. The easiest way to illustrate this process is to describe the way in which a new transaction is added to the ledger. The initial step is called endorsing. In this phase, the peers required for the specific transaction are required to digitally sign the transaction in order to agree that it should be processed. The number of endorsements as are necessary for the transaction, as well as what peers are required to endorse is defined in the consensus model. The transaction is then sent to the ordering nodes of the network which run a pBFT algorithm to find consensus on the order of the new incoming transactions. As the third step, when the ordering nodes reach consensus, the transaction is broadcast on the channel, where peers run a validation on the transactions to verify that the ordered transactions are correct, before appending them to their ledger.

Value on the system

There are no inherent cryptocurrencies in the design of the Hyperledger Fabric framework. It is possible to create cryptocurrencies and token applications on the platform through the use of chaincode.

Interacting with the system

Each shared ledger in a channel has a chaincode defining the operations available to modify the state of the ledger. Modifying the state creates a transaction which will pass through the consensus algorithm for the transaction to be endorsed, ordered and validated by peers in the channel.

A new transaction, intended to update the ledger state, is constructed in a specific way. The transaction contains a header with data about the chaincode and the chaincode version, a signature of the client application, the proposal as the input parameters to the chaincode creating the proposed ledger update, the response as the before and after the value of the proposal, and lastly endorsements of the transaction.

A query for information about the current state of the ledger can be sent to the peer node which holds the database with the current state. This is considered a read-only transaction and is therefore not required to broadcast as a data transaction of the ledger.

Access and privacy

In order to ensure that a blockchain network is permissioned, there need to be components to manage network identities for the network's organizations. A membership service provider (MSP) is the component handling this in Hyperledger Fabric. A MSP is an abstraction of a variety of access protocols and is installed to each node in the network. It can then operate as a validator for incoming transactions to the node. By the use of different cryptographic protocols, an identity can be connected to an organization and thus be granted access to the node.

4.1.2 System performance

Hyperledger Fabric is a modular blockchain framework and has not, according to IBM, yet been performance-tuned and optimized (Androulaki et al., 2018). Due to the modularity of the framework, the system performance depends on a number of parameters. These parameters include the choice of application, transaction size, the ordering service, the consensus implementation, the number of ordering nodes in the network, the hardware on which the nodes run, and further configuration parameters.

Computing performance

IBM has studied the performance of the Hyperledger Fabric framework by evaluating it with respect to the number of transactions that can be performed per second (Androulaki et al., 2018). However, the performance is affected by several factors, and due to the modularity of the framework, separate networks can be designed and optimized for different purposes. Therefore, different Hyperledger frameworks can have varying performance seen to transaction throughput etc.

The application created to study the performance was designed as a UTXO cryptocurrency using a key-value database. Nodes were running on the Hyperledger Fabric version 1.1 instrumented for performance evaluation, hosted in IBM Cloud virtual machines interconnected with 1 Gbps networking. The nodes were 2.0 GHz 16-vCPU with 8 GB RAM and SSDs as local disks. The network had three ordering nodes, and five peer nodes, all being endorsing peers. Separate virtual machines ran all nodes.

The factors that shifted were the block size and the number of CPU for each peer. Block size of between 0.5 MB, and 4 MB was considered, and the result showed that the network managed over 3000 transactions per second for block sizes over 2 MB. Smaller blocks had a lower throughput and bigger blocks slightly higher. The end-to-end latency was continuously increasing with greater size of the blocks.

The block size was set to 2 MB when testing the effect of the number of CPUs for each node. The result showed that increasing the number of CPUs affected the transaction speed positively. The transaction speed was just over 1000 transactions per second when running the nodes on 4 CPUs. The corresponding number for 32 CPUs was slightly below 3500 transactions per second.

System stability

The stability of the developed network is directly connected to the stability of the participating peer nodes and ordering nodes. Increasing the number of nodes would increase the fault tolerance and thus the system stability. The stability of the system can vary between different networks due to the networks' different setups. The fact that the network is permissioned gives the creators increased influence over the system stability. However, the default consensus algorithm, pBFT, entails a fault tolerance of $\frac{1}{3}$ (Bach et al., 2018).

4.1.3 Platform ecosystem

Hyperledger Fabric is part of the Hyperledger greenhouse. The greenhouse is a collection of frameworks and tools to let blockchain technologies and ideas co-evolve and emerge. While there is a focus on solving problems with enterprise implementations, the code is open source and available to anyone for usage and contribution.

Programming possibilities

Hyperledger is, as previously stated, a framework and serves only as a foundation for developers to create their blockchain solutions. Consequently, the framework gives the developers full control of how the system is implemented and how it will operate.

Chaincode is written in one of three general-purpose programming languages: Go, node.js, or Java, and runs in a docker container in order to separate it from the peer it is executed on. The state created by the chaincode can only be accessed by that contract. Contracts can, however, be given access to other contracts which means they can access their state through the given transactions.

Developer community

Hyperledger provides not only the possibility for developers to contribute to the open source code, but also a vast guide about how to both understand the key concepts of Hyperledger Fabric, but also about how to create applications. Hyperledger encourages users to help develop the training material, spread the word about Hyperledger, and to engage in meetups on different locations and times. The organization refers to their code of conduct describing accepted and acceptable behaviors to promote high standards of professional practice. The official GitHub repository of Hyperledger Fabric consists of 9090 commits from 168 contributors (Hyperledger, 2019).

User access

Data and information from the ledger can be accessed from outside of the network by a client application. The client application is connected to an organization and can connect to one of the organization's peer nodes which will provide ledger access and information through the chaincode transactions.

Hosting costs

The Hyperledger Fabric framework does not specify any fees for computation through deploying, ordering or validating chaincode.The endorsing nodes and ordering nodes participating in the consensus mechanism are not incentivized by on-chain tokens or cryptocurrencies, but rather to avoid off-chain costs connected to malicious behavior. Because of this, transaction and execution do not have to exist if not desired. Instead, the costs for an application being developed is consisting of both the writing of the chaincode and the configuration of the system itself. When developed, the creators also need to provide their infrastructure for running the system. Every participating node needs to set up their peer node, and endorsers and orderers need to be compensated outside the system.

Areas of usage

The solution of Hyperledger Fabric is mainly intended for business-to-business interactions. Use cases, according to IBM, include areas such as trade logistics, dispute resolution, foreign exchange netting, contract management, food safety, diamond

provenance, rewards point management, identity management, low liquidity securities trading and settlement, and settlement through digital currency (Androulaki et al., 2018).

4.2 Neo

Neo is an open source blockchain project with the purpose to establish a platform for managing all types of assets, through the use of digital assets, digital identity and smart contracts. The digital assets represented in the system are of two types. Global assets are exchangeable on the entire platform, whereas contract assets require access to the contract the asset is issued on. On Neo, anyone can issue both global and contract assets.

4.2.1 System design

Neo's blockchain structure is consisting of a hash linked list of transaction blocks as described in 2.1.1. The system does use a transaction model with account balance redundancy to model the state. The consensus mechanism, dBFT, is based on the election of validator nodes.

Data structures

Neo relies on a blockchain architecture that is in many ways similar to other known blockchains. Neo does, in addition to the UTXO model, use account states to model the state. The UTXO model is used for global assets, whereas the account model is primarily used for contract assets. Note that there are no additional data structures connected to the block used to record account states. Thus, no mapping between accounts and their balances are recorded systematically in the blockchain. Instead, validator nodes need to maintain data sets with states and contracts separately.

Participants

There are two types of nodes defined in the protocol. Peer nodes, that can broadcast, receive and transfer transactions or blocks, and validator nodes that can create new blocks. Peer nodes accept the state of the ledger produced by valid transactions signed by validator nodes. These bookkeep transactions in order to maintain the ledger.

Consensus mechanism

Neo uses a consensus mechanism called dBFT, as described in the background chapter. As other consensus mechanisms, it can be divided into a consensus model and a

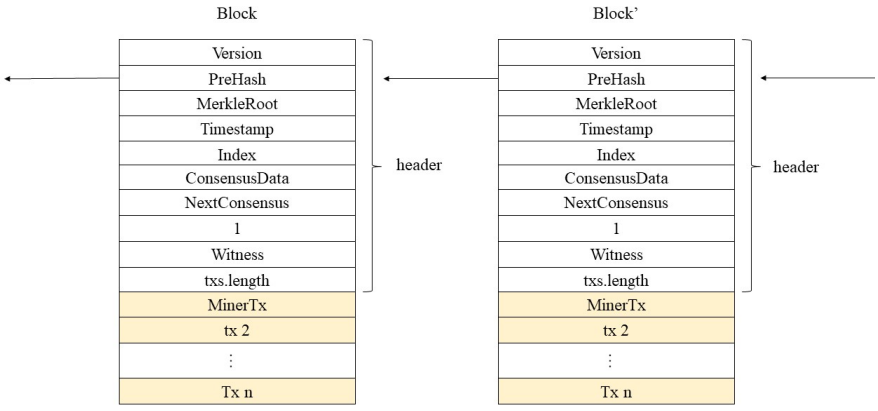


Figure 4.4: Modelling of Neo's blockchain implementation (Interpretation of figure from Neo Organization (*NEO Documentation*, 2019))

consensus algorithm respectively.

The decision model consists of an election of validating nodes. NEO holders that have a stake in the network decide upon several validator nodes, and a specific list of validator nodes, i.e., nodes that gets the privilege of producing and validating new blocks. This voting is connected to holders public addresses and recorded in every block.

In Neo's consensus algorithm, the elected validator nodes take turns to produce the next block. Sequentially, the other validator nodes make a majority decision on whether to accept the block or not.

Value on the system

The economic structure on the network is based upon the two global assets NEO and GAS. NEO can be seen as stakes in the platform, which gives its holders mandate to manage the network and for example vote for validator nodes. NEO holders get shares of the other global asset, GAS, as it is continuously issued, and also gets paid in GAS when assets are registered on the network. The smallest amount of NEO existing is 1 NEO, and it cannot be further divided. GAS is used to pay for system operations, i.e., compensating validating nodes for their validating. This whole structure, in theory, works to hold down transaction costs. If higher transaction costs are induced, fewer actors will register assets on the platform. Hence, NEO holders will get less compensation for assets registered. This gives NEO holders the incentive to vote for low-cost validators, who will yield them more value.

Interacting with the system

In Neo, transactions are the way for individuals and smart contracts to operate with the system. To be executed, a transaction needs inputs and outputs, to signal which UTXOs that are going to be used and produced respectively. There are nine transactions defined, according to Table 4.1.

Transactions in the Neo Protocol		
Transaction	**System Fee**	**Description**
MinerTransaction	0 GAS	Assign byte fees
IssueTransaction	0-500 GAS	Inssuance of asset
ClaimTransaction	0 GAS	Assign GAS
EnrollmentTransaction *	1000 GAS	Enrollment for validator
RegisterTransaction *	10000 GAS	Assets register
ContractTransaction	0 GAS	Contract transaction
PublishTransaction *	500 GAS	Special Transactions
InvocationTransaction	0 GAS	Special transactions

Table 4.1: Transactions and fees in NEO, interpreted from (*NEO Documentation*, 2019). Transactions marked with (*) are deprecated and replaced with system operations. Fees are still the same for corresponding operations.

Access and privacy

The Neo network is accessible for use by anyone that wants to participate. To take part, only creating an account represented by a public key is needed. Neo's intention is to let participants get transactions prioritized through use of public identity, and if there is a wish to be anonymous, one can pay transaction fees to be prioritized instead. However, to be part of the validation process, nodes need to be elected by NEO holders. For individuals, this process is similar to a political election, since candidates need to identify themselves and argue why they should be elected.

4.2.2 System performance

This section intends to give insight to the system performance of the Neo platform by analyzing several different component. All things considered, Neo's consensus mechanism makes the system efficient in terms of transaction processing. However, a high degree of centralization makes the network reliant on a few numbers of validator nodes.

Computing performance

The design parameters of its consensus mechanism makes Neo's validation process very effective seen to transaction processing time, and at the same time more efficient in terms of energy consumption than a proof of work algorithm. The Neo network today is taking 15 to 20 seconds to generate a new block. It can process around 1,000 transactions per seconds, and 10,000 transactions per second might be achievable with proper optimization.

System stability

The consensus algorithm entails that the fault tolerance f of the network is

$$f = \lfloor (n-1)/3 \rfloor$$

where n is the total number of validator nodes, since the validating nodes always choose one single valid block to continue building on. The Neo blockchain is also characterized by an attribute called immediate finality. As long as the fault tolerance is not exceeded there is consensus on exactly how the entire blockchain looks.

Currently, there are only seven validating nodes. Out of these, only one is controlled by an actor not directly related to the Neo organization. The reliability of Neo is highly connected to the validating nodes and these therefore have a public identity. This is to facilitate legal measures if some of the nodes would act maliciously.

4.2.3 Platform ecosystem

To make Neo accessible for developers, it has been made compatible with several mainstream programming languages. The organization behind the platform strives to establish a strong community around it, which however is still in its early stages.

Programming possibilities

The Neo network enables developers to code in their own preferred languages. Languages such as C#, Java and Python can all be used for coding smart contracts, which is intended by the original creators to be accelerating for the network since programmers do not need to learn a new language.

Developer community

Neo is an open source development project, and the organization expresses a vision of building a strong development community. The community provides several tools and softwares, such as development kits and smart contract compilers, planned on being supportive for developers to maximize the potential of the development ecosystem. As

of May 4th, 2019, the official GitHub repository of Neo has 32 contributors of which the founder, Erik Zhang, is the biggest seen to lines of code (Project, 2019).

User access

The Neo organization supports different types of node programs, where participants can either be full nodes or light nodes. There are both desktop, and mobile clients available for download, which do not need to store the entire blockchain to function. Hence, the only thing needed of users is to create an account on the platform. With access to the contract address they are ready to interact with the application.

Hosting costs

The Neo network is an already established network, meaning a developer of an application can deploy contracts without needing to set up an entire infrastructure. Development costs are therefore solely consisting of the costs for developing the smart contract.

Validating nodes do get compensated with transaction costs. However, their public identity also incentivizes them to behave honestly since legal measures can cost them outside the network, and their reputation can be damaged. Transaction costs are zero as default, and every block contains a maximum of 21 free transactions. Users are though recommended to pay transaction fees to be prioritized by validators.

Deploying a contract on Neo comes with a fixed cost and is the most expensive cost when launching contracts on the platform. One single contract deployment implies a cost of 500 GAS. Operation costs, however, are cheaper. The fee for writing to the account specific, persistent storage is set to 1 GAS per KB, and a get-operation costs 0.1 GAS. Since the initial 10 GAS of every smart contract execution is free, a lot of transactions can be done for free on the network.

Areas of usage

The Neo network has as already mentioned an outspoken focus on digitizing assets. Its compatibility with PKI (Public Key Infrastructure) X.509 standard lets users and asset issuers provide their identity if necessary, enabling registration of both digital and physical assets.

Of the applications existing today, most are focused on B2C solutions. Examples are a music trading platform, a peer-to-peer shipment solution, and various saving and general purpose trading platforms.

4.3 Ethereum

Ethereum is, as stated by the Ethereum foundation, ”a decentralized platform that runs smart contracts”. It is designed to facilitate the development of decentralized applications, in which smart contracts are the core. In the Ethereum context, a smart contract is most often related to some type of economic transaction, mainly of its own cryptocurrency, Ether. One common abbreviation is EVM, short for Ethereum Virtual Machine, which is used for referring to the system when thought of as a computer, responsible for executing contracts and computing the system state.

4.3.1 System design

Ethereum exclusively uses an account model to represent the state of the network. The consensus mechanism is based on a PoW algorithm and thereby working without the need of trust in any individual in the network. This enables total anonymity on the platform if desirable. The global currency Ether is a central part of the system, usable both to transfer value between peers and to pay for system operations.

Data structures

The Ethereum blockchain state is based on accounts which are divided into two types, externally owned accounts (EOA), and contract accounts. The EOA:s are controlled via users' private keys, whereas contract accounts can be seen as autonomous agents able to act without any user input. Contract accounts are controlled by their contract code and can only be activated by a transaction call from an EOA. All contracts can hold value.

Ethereum uses multiple tree structures connected to each block. Besides the patricia merkle trie used for storing transactions, the most important one to note is the patricia merkle trie used to store accounts and their balances. It lets nodes perform vastly swifter validations

Participants

There are three types of roles on the Ethereum network. Full nodes, meaning a client that stores the entire network. Miners, that are full nodes but also participate in the process of validating blocks. Finally, there are light nodes, that only store parts of the blockchain relevant for them. Light nodes are commonly serving as user clients, since their small storage consumption enables, for example, running on a smartphone.

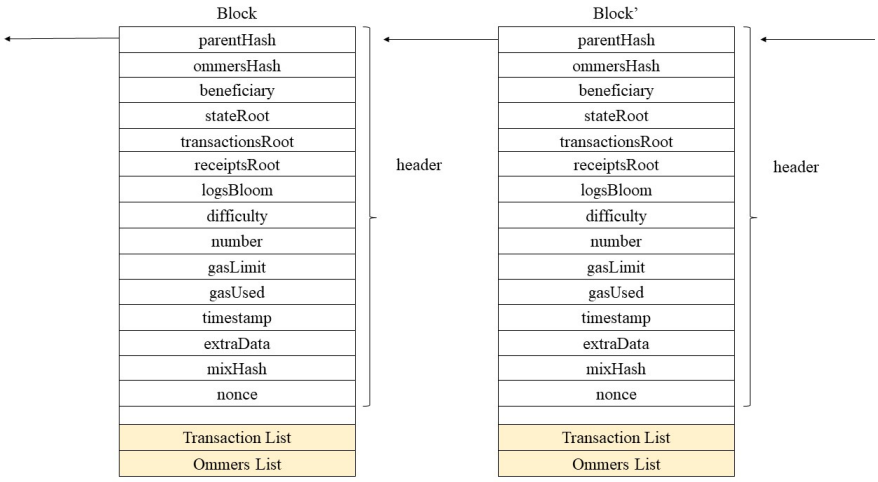


Figure 4.5: Modelling of Ethereums's blockchain implementation (Interpretation of Ethereum Yellowpaper (Wood, 2014))

Consensus mechanism

Ethereum uses a decision model functioning without trust in any specific party, obtaining consensus through the choice of the blockchain branch that has got most work put down on it. This is enabled through the use of a PoW consensus algorithm.

Ethereum is using a PoW-algorithm called EtHash. Ethereum's algorithm lessens the advantage of using application-specific integrated circuits to speed up their work and hence rewards also smaller miners. Optimized for short block times, the probability of two miners finding a new block at the same time is high in Ethereum. To handle this, Ethereum uses a protocol called Greedy Heaviest Observed Subtree (GHOST). This is a way of, by incentive, getting miners to work on the main chain to a larger extent.

The Ethereum foundation has expressed a wish to switch to a PoS consensus algorithm, but no clear date has been set or outspoken. This needs to be taken into consideration when developing applications on the platform since it can lead to dramatic changes in system behavior or even hard forks dividing the entire network in two.

Value on the system

There are two ways of transferring value on the Ethereum platform. First, the global currency Ether can be used to make payments peer-to-peer both directly and through smart contract execution. Second, own tokens can be issued through smart contract implementation. The key difference is that own tokens can only be used within the

context of the defined contract, whereas Ether can be used globally on the network. Miners are paid with Ether. However, transactions and execution on the system come with fees set in a value unit called gas. Gas payments are elaborated in the following paragraph.

Interacting with the system

The term transaction is in Ethereum referring to a signed data package storing a message being sent from an externally owned account to another account. Another term in Ethereum is a message, which is similar to a transaction, with the difference that it is sent from a contract account. Both transactions and messages can contain data, allowing them to transport instruction sets back and forth to contracts. Since the instructions can be more or less computationally heavy, the sender needs to specify a startgas value, i.e,. a limit for how much gas a transaction is allowed to consume, and a gasprice value, which is the amount of Ether offered per unit of gas to miners for carrying out each computational step. Computational steps have their prices in gas, making the cost analysis dependent on which specific operations are executed.

There are two types of transactions on the Ethereum network. A contract creation transaction is used to create an account with its associated code. A message call transaction is used to interact with the system and for example, make it execute smart contract scripts. The latter transaction type requires a data field containing the inputs that are going to be used in the message call. This can consist of different system operations requested to be executed. System operations can be ultimately divided into computational steps. Every transaction costs 21000 gas, and in addition, each computational step costs 1 gas. This is consequently the price of simple EVM operations such as simple addition, consisting of only one computational step. All operations available are listed in the Ethereum Yellowpaper, together with their gas consumption.

Access and privacy

Ethereum is an open source, public blockchain platform, meaning anyone can theoretically participate with any role in the system. The consensus mechanism is designed to let all parts be anonymous. Even though the system works with identification with public keys only, more identification can be included in the scripts of smart contracts.

4.3.2 System performance

Ethereum has a short block time and high transaction throughput for using a PoW algorithm. However, the platform is subject to different security threats, awaiting large scale protocol changes, and its degree of centralization in practice has been questioned.

Ethereum TPS calculation	
Block gas limit	8000000
Transaction costs	21000
TPB	380.95
Block time	13.13
TPS	29.01

Table 4.2: Snapshot of TPB and TPS in the Ethereum network on May 3rd, 2019 (Etherscan, 2019)

Computing performance

The transaction throughput on Ethereum can be calculated through the use of some simple calculations. First, the block gas limit is divided by the transaction cost to get the number of transaction per block (TPB). Then, the TPB is divided by the block time to get the transactions per second (TPS). Table 4.2 specifies the results of such calculation, with block gas limit, and block time retrieved on May 3rd, 2019 (Etherscan, 2019). The transaction cost is fixed and specified in the protocol (Wood, 2014).

System stability

Ethereum is subject to the already known 51 percent attack related to the use of PoW, and are, as described in the background, at risk having transactions tampered with malicious miners controlling 25 percent of the computational power. Ethereum is also exposed to others because it includes a turing-complete programming language, meaning it can carry out any conceivable computation. Since this entails a possibility to run loops, there is a risk for running infinite ones. This could be used for emptying accounts since such loop would run until the gas balance is empty. For operations, however, a max gas amount is set to avoid such scenarios.

Since the validation process of Ethereum is open for everyone, and not reliant of identification and trust in individual miners, it is highly decentralized seen to its number of miners. Even though Ethereum has a higher network latency than the commonly known Bitcoin network, indicating larger geographical distribution among miners in the network, the degree of actual decentralization is however discussed (Gencer, Basu, Eyal, Van Renesse, & Sirer, 2018). Even though most of the biggest miners are mining pools, it has been argued that the ones in charge of those still possess serious amounts of power (Gencer et al., 2018). To see how the distribution of computational power is throughout the Ethereum network, one can use (Etherscan, 2019), where mining power measured in hashrate, i.e., how many hashes every miner solves per time unit.

4.3.3 Platform ecosystem

Developing on Ethereum recommends the use of an own programming language for writing smart contracts. The platform is surrounded by a strong community that provides useful tools for developers. The infrastructure around the platform makes it easily accessed for application users.

Programming possibilities

Solidity is an object oriented programming language designed specifically for smart contract coding on Ethereum. Smart contracts are recommended by the foundation to be written in Solidity. Even though it is a new programming language, it has a lot of commonalities with JavaScript.

Developer community

The founders are still important influencers and contributors to the system, but as of May 4th, 2019, 407 individuals in total have contributed with 10880 commits to the Go implementation of the protocol in the official Ethereum GitHub repository (*Ethereum*, 2019). Illustrative of the influence from the community is also the fact that the Ethereum foundation is referring to community-sourced documentation for detail and explanation about the platform.

The Ethereum community has also provided sites such as Ethereum Stack Exchange, which is part of the Stack Exchange network. In total, there exist 25000 questions on the Ethereum Stack exchange. Furthermore, through use of a few open source tools, all produced by the community, it is possible to set up one's development environment for running and testing smart contracts, both locally and on test networks, destined for the Ethereum network.

User access

Theoretically only an EOA, the application binary interface and the address to the contracts belonging to the application, is needed for a user to interact with any application on Ethereum network through a node. In practice, to enable mainstream user adoption, a minimum of a user client interface, is needed.

Hosting costs

Developers can deploy application contracts to the already existing Ethereum network, meaning development costs are related to smart contract creation and deployment only.

The miners securing the network does, however, charge both transaction and execution costs, which need to be considered when designing applications on the platform.

Ethereum system operations are paid for with a value unit called gas. The gas amount required for different operations is fixed. However, the gas itself is bought with Ether, which value fluctuates in relation to fiat currencies. To calculate costs of transactions and operations in fiat currency, the current gas price and the Ether price can be fetched from a blockchain explorer such as (Etherscan, 2019).

First, every transaction comes with a fixed cost of 21000 gas. A contract creation operation is set to 32000 gas. The actual cost of deploying a contract is however higher, since it has to be executed to be fully deployed. Examples of execution costs are a load from the persistent storage, that costs 6400 gas per KB, and writing to the storage, that costs 640000 gas per KB.

Areas of usage

Most applications built upon the Ethereum platform can be roughly divided into three areas of usage. Finance, including platforms for derivative trading, hedging and other. Semi-financial, with the best example being the one of leasing out computing capacity in exchange for cryptocurrency. The last one is the one of autonomous organisations, such as voting systems. All of these either have viable solutions up and running today, or have proven to be theoretically viable.

4.4 Synthesis

In Table 4.3, descriptions of the three platforms are summarized and presented beside each other. This table highlights key takeaways from this chapter and can be used to unearth differences between platforms, thereby laying the groundwork for discussion in Chapter 6.

Property	*Hyperledger Fabric*	*Neo*	*Ethereum*
System design			
Data structures	Optional	UTXO model and account model coexist	Account model
Consensus mechanism	pBFT	dBFT	PoW
Value on the system	Possibility to implement token system	On chain currencies and tokens	On chain currency and tokens
Interacting with the system	Transactions to modify state, queries are not recorded as transactions	Transactions to do all types of operations with system	Transactions to do all types of operations with system
Access and privacy	Blockchain permissioned, validators specified in system configuration	Blockchain open, validators elected	Blockchain open, open validation participation
System performance			
Computing performance	Around 3000 TPS, immediate finality	1000 TPS, immediate finality	29 TPS, probabilistic finality
System stability	High degree of centralization, risk of system disruption with 1/3 malicious nodes.	High degree of centralization, risk of system disruption with 1/3 malicious nodes.	Low degree of centralization, risk of forged transactions with 1/4 malicious nodes

Continued on next page...

Property	*Hyperledger Fabric*	*Neo*	*Ethereum*
Platform ecosystem			
Programming possibilities	Developer can implement system in preferred coding standard	Compatible with mainstream coding languages	Own programming language, influenced of JavaScript
Developer community	168 contributors on GitHub, community-sourced documentation and development tools	32 contributors on GitHub, community-sourced development tools	407 contributors on GitHub, community-sourced documentation and development tools
User Access	Permissioned, requires network-adapted client application connected to participating organization	Open for everyone, facilitated with application specific user interface	Open for everyone, facilitated with application specific user interface
Hosting costs	Off chain costs, primarily of development and establishment of system infrastructure, transactions and execution are free on chain.	On chain costs, primarily for deployment, but also for transactions. Paid with global currency, priority trough identification or payment.	On chain costs, deployment and transaction costs. Paid with global currency, priority through payment.
Areas of usage	Business-to-business use cases	Asset registration, financial applications, social media, betting etc.	Autonomous organizations, financial applications, social media, betting, etc.

Table 4.3: Summary of platform examination conducted in Chapter 4

Chapter 5

Development

The platforms investigated in chapter 4 differentiate themselves from each other in several aspects. However, it is hard to draw any conclusions regarding which of these differences that are of importance. This part of the study aims to solve this by developing a smart contract for a ROSCA system, described in 2.4. By creating an application using the technology, an insight of which properties are of importance could be obtained. The insight could then combined with the platform examination to determine which properties are decisive and non-decisive in the context of a ROSCA.

The first section of this chapter explains why the choice blockchain platform fell on Ethereum to support the smart contract. The second section describes the development process and motivates the decisions made, leading up to the final contract with its functionality and limitations. The last section presents the obstacles encountered during the development process, dissecting them in order to gain further knowledge of their significance.

5.1 Selection of platform

Being limited to developing the ROSCA application on a single platform, the study demanded making a platform choice, which the problem statement of this study claims difficult. The choice fell on Ethereum for hypothesized fitness based on the initial platform examination, the convenience of the study, and suitability for the ROSCA application.

System design

One reason was a strive of developing an application on an as decentralized network as possible. This would eliminate the need of a trusted party to operate the ROSCA, and thus the possibility for that party to be held responsible in the case an error occurs. On Hyperledger Fabric, validating the state is done by a small set of selected actors, and on Neo, it is currently done by seven validator nodes. However, on Ethereum the validation of the state is shared by several thousand users.

Another reason was that the value transactions should be attained on-chain. Direct payments through other media would need validation and be processed to be recorded on the chain. On Neo and Hyperledger on-chain value transactions would be more complicated to achieve. On Neo smart contracts do not have the possibility to hold any global cryptocurrency within the contract. On Hyperledger there are no inherent

cryptocurrencies or tokens that can be utilized. In contrast, Ethereum has Ether, which is a global token that smart contract have the possibility to hold.

System performance

Furthermore, each contract represents a single ROSCA, meaning a platform with low deployment cost is advantageous. In contrast to Ethereum, Neo has a substantially higher deployment cost. Hyperledger does not implement any deployment fees. However, it has substantially higher costs in terms of development and implementation, since it only provides a framework. Similarly, the ROSCA did not require immediate finality to function. This made platforms which do not have immediate finality, a viable choice, whereas other applications might view this as an issue.

Platform ecosystem

Because no prior knowledge related to development on these platforms existed within the team, the development of a smart contract called for a great quantity of information gathering. Consequently, the development community and documentation related to the platform played a decisive role. Information related to Ethereum was much more comprehensive which the initial platform examination showcased.

5.2 Development environment

This section intends to give insight into the development process of the project, consisting of two parts. The first phase looked at setting up an appropriate programming environment. This was a prerequisite for writing the smart contract, which was the second part of the development process.

5.2.1 Environment setup

The first step when developing an application on any platform is to set up a programming environment. This was planned to serve different phases of development like writing, compiling, debugging, testing and deploying of application software. Two different development environment setups were used by different developers. One of these is the Integrated Development Environment (IDE) called Remix. It is a web-based environment which enables developers to write, compile, debug, test and deploy contracts to Ethereum networks. To test a contract, either software defined unit tests, or manual testing could be used. For manual testing, the contract had to be deployed to an Ethereum network, or a local runtime simulating the blockchain. Remix implements a local runtime in Javascript which was used during short development iterations while it also may connect to a full node through remote procedure calls (RPC) to an Ethereum node.

When connecting to a full node through RPC, the full node could either be part of one of Ethereum's test networks or the Ethereum main chain. After the contract was created and tested locally, it was deployed to one of Ethereum's test networks called Rinkeby, allowing interaction with the contract from several physical peers. As a developer, a choice between running a full node or connecting to an external node had to be made. As running a full node is more demanding, the latter option was selected, offering instant access making it possible to focus on development. This external Ethereum node was hosted by Infura (Infura, 2019), which provides access to Ethereum nodes through RPC as a service.

Another option explored was a terminal-based setup using official and third-party software to perform different actions. This required a text-editor for writing contracts and the official solidity compiler `solc` for compiling. For debugging, testing and deploying contracts, truffle was used. Truffle is an open-source tool part of the Truffle suite created for Ethereum development. Like Remix, it implements both a local runtime for testing purposes as well as an RPC interface for connecting to full nodes for live network deployment and interaction. This development environment was set up but not utilized, since the ease of access of the Remix IDE made it sufficient for exploratory purposes in the small scale setting.

5.2.2 Reference documentation

Relative to setting up the programming environment, most time of the project was spent on developing the contract. The developmeant process resulted in a contract with limited functionality which was deployed and tested on the Rinkeby test network.

The initial phase of contract development was to get acquainted with Solidity as a smart contract language. This was done by reading documentation from the Ethereum organization such as (Ethereum, 2019), which gave sufficient knowledge of the language for writing simple contracts. The documentation gave the developers insight concerning the structure of contracts. For instance, lots of example-code illustrating good structure was available on the organization's homepage as well as documentation regarding coding conventions. Furthermore, it provided a considerable quantity of documentation related to security which highlighted a list of security considerations. Lastly, the online communities surrounding Ethereum proved to play a crucial role when unexpected problems arose, primarily when no sufficient documentation regarding the obstacle existed. Several issues related to the contract was resolves leveraging online communities where common issues were discussed. These discussion boards highlighted suggested solutions which accelerated the development process of the contract. After learning the basics of contract development by creating simple examples, the development of the ROSCA application began.

5.3 Application product

The final code of the developed contract can be found in Appendix B. While the contract is only one part of a usable application, the development disregarded user client development as the smart coordinates the peer to peer interaction, implementing the basic design functions of a ROSCA, as described in section 2.4.

5.3.1 Contract data

In the contract code, variables storing data regarding the individual participants, and the circle as a whole are declared. Participating peers are represented as an object of multiple data points. This object, called `participant`, contains the student name, student identification and the student's current department of study. It also contains information of how much Ether the student has deposited to or received from the contract as well as the bid placed by the student in the ordering auction.

Several participants may be stored in a map called `participants`, which is a collection of key-value pairs, meaning each participant can be referenced using a unique Ethereum address. In Ethereum's virtual machine, it is not possible to iterate through this type of collection, which requires addresses of participants to be stored redundantly. The ability to iterate through each address is necessary for certain transactions like `payOut`.

5.3.2 Contract transactions

An instance of this contract represents a saving circle between several individuals and encapsulates all on-chain transaction made in the saving circle. Creating an instance is done using the constructor of the contract. The parameters of an instance are passed when constructing the contract which are `amount` `interval` and `startDate`. When an instance has been created, users can interact with it by performing transactions which invoke the functions defined in the contract, changing the state of the contract instance on the Ethereum blockchain.

Joining the circle

Instead of letting the contract does collect a list of participants when initializing the contract, joining a circle is done using an opt-in model. This is done by the user invoking `joinCircle`, which appends the personal details mapped to the participant's public address to the collection of participants. This function can be called using any public address, and the personal details of the participant are not verified in this implementation.

Rotation

Since rotation was implemented using the Vickrey auction, the contract implements this bidding logic. The `placeBid` function may be invoked by participants who wish to partake in the auction, which changes the bid variable in the corresponding participant. In contrast to the last function, only students who have already joined the circle can call it, which is made possible by the use of the utility function `isInCircle`. This function evaluates if the public address of a participant is stored in the collection `participantAddresses`.

Transaction of value

The transaction of value is done using native Ether transactions through the function `pay`. In addition to that, the address of the payer needs to map to a `participant`. It also requires the amount transferred to be equal to the set amount of the circle. If these requirements are satisfied, the variable `hasPaid` for the corresponding participant is changed to true. The contract models the way the third party collects value from participants before paying to a participant.

The function `payOut` invokes this payment from the contract to the winning participant. After checking if the current time surpasses the time of the disbursement, the function calculates the amount to sent back to the winner of the auction. This is equal to the Ether the contract holds, minus the second highest bid, according to the principles of a Vickrey auction. After that, the remaining amount is shared equally and paid out to the remaining participants.

Penalty

In this contract, there is no logic to support the case where a participant defaults. As a ROSCA traditionally relies on social capital to punish participants who do not pay in time. This type of functionality would also require some verification of personal details participants provide when joining the circle.

5.4 Obstacles encountered during development

During the development of the application, several obstacles where identified. One of these was regarding the possibility to hide data on a public blockchain such as Ethereum. In the contract, information was stored in contract variables, accessible to nodes. Consequently, the bid was not hidden, and therefore the Vickrey auction had some glaring shortcomings. By leveraging the development community, a possible solution Awas found in that cryptography could be used to overcome this obstacle. By practically implementing the solution, knowledge regarding the possibilities of hiding data on a public blockchain was gained. Another obstacle encountered was

regarding how the contract could be optimized to lower operational costs. However, to answer this question, an understanding of the Ethereum cost structure was necessary. To analyze it, the developed contract was utilized. The analysis pointed out that transaction costs are highly dependent on how much data that are stored in permanent storage.

5.4.1 Hiding data on a public blockchain

One challenge the group encountered was regarding the bidding function of the smart contract. In the contract, each participant placed a bid which was stored in the contracts public storage. This design has its obvious drawbacks such as the bid were not hidden from each participant. The reason the information is not hidden from the other participants is that transactions and information stored in storage, are public as well as permanent. By leveraging the development community, a possible solution was found where cryptography could be used to solve this dilemma. Instead of each participant placing the bid in plain text, they submit its corresponding hashed value. When the bidding period has ended, each participant deposits the money and reveals their bid by sending it in plain text. The bid is then hashed and compared with the stored hashed bid they sent in beforehand. Ethereum offers some utility functions that exist in the global namespace, meaning that they are visible throughout the contract.

An implementation of this was constructed to gain further understanding concerning how cryptography could enable a hidden bid in a public blockchain such as Ethereum. By practically applying this solution, insights were attained regarding its applicability as a solution for privacy on a public blockchain. The contract can be found in Appendix C. Similarly to the contract representing the saving circle, this contract contains similar state variables. It includes participants mapped to their corresponding address where each participant has variables representing the secret bid they have placed. The function `placebid` is invoked when the participant places a secret bid, which represents the hashed bid. The secret bid is stored in the participant object, which is mapped to the same address as the one who invoked the function. The function `reveal` is then used for the participant to reveal his bid. It takes in the variable `bid`, which represents the numeric value of the corresponding bid. The variable is hashed by utilizing the hash function available on Ethereum and compared to the corresponding stored secret bid. If these values match, it is deemed to be the secret bid that the participant placed earlier.

A problem that occurred when constructing the `reveal` function was how to cast the type of the `bid` variable to a hashable value. However, by utilizing the Stack Ethereum Exchange the function `toBytes` was constructed which solved this dilemma. The value of `bid` is now compared to the variable `highestBid` which gets overwritten if it is of higher value. Finally, the participant's address is stored in the variable `highestBidder`.

A limitation of how this contract implements cryptography to enable hidden bids is that there is always a risk of a malicious user decrypting it by continuously hashing

different numeric values until a match with the secret bid is obtained. In the contract, the range of possible bids a participant places has to be big enough so no malicious user could figure it out by continuously guessing. However, this is a significant limitation of the contract. Consequently, the cryptography implemented in the contract does not fully solve the fundamental dilemma of privacy on a public blockchain.

5.4.2 Operational cost analysis

A cost-effective contract is highly dependent on the design of the contract. The reason for this is that the contract design affects the computational work, on which the cost is based on. To gain an understanding of how the contract could be designed to lower the cost, one has to understand what causes them. Consequently, an investigation of the associated fees related to the saving circle constructed earlier was conducted. This, by leveraging the Remix IDE debugger. The debugger was used to collect data regarding the executing and transaction costs but also to identify the most expensive operation relative to the execution cost for each function. By compiling the developed contract in the Remix IDE, the operational costs were measured in GAS. These were measured both per invocation and per operation, presented in table 5.1 and 5.2 respectively.

Cost structure of each function		
Function	*execution cost[gas]*	*Total cost[gas]*
Constructor	61 264	1 444 888
JoinCircle	31 426	56 602
pay	21 644	42 916
placeBid	21 265	42 729
pay	31 487	52 759

Table 5.1: Measured execution cost of implemented functions.

Table 5.1 shows the execution and total costs for each function that is invoked by the users. The data regarding the costs were gathered by conducting tests on the contract in Appendix B. The execution cost is related to the cost of performing all the corresponding operations in that function. Each of these operations relates to a specific amount of work. However, the fee is not only determined by the operations of the contract but also by the interaction itself. For each transaction, a base amount of 21 000 gas is charged. Furthermore, costs are added depending on the amount of data the transaction contains. A transaction that creates a contract costs a base amount of 32 000 gas and additionally 200 gas per byte of the source code. The table specifies the cost in gas instead of Ether due to the amount of fiat currency the miner acquire depends on the gas price.

Table 5.2 shows which operation is the most expensive one for each function. The data was acquired trough letting all the operations carried out by each function be analyzed, once again of Remix IDE debugger. The cost presented is obtained by taking the accumulative fee from each operation. The debugger only takes into consideration

Operation relative to the execution cost		
Function	*Operation*	*cost[%]*
Constructor	SSTORE	98
JoinCircle	SSTORE	85
pay	SSTORE	92
placeBid	SSTORE	94
pay	SSTORE	48

Table 5.2: Most expensive operation in implemented functions

operations related to the execution of the contract. Therefore, the cost in the table is presented in relationship to the execution cost and not to the transaction cost. Another important thing to highlight is that the cost related to the transaction, and not the execution, mostly depends on the base amount of 21 000 gas that each transaction causes. This is derived by taking the difference between the transaction and execution cost for each function. The difference amounts to around 21 000 to 25 000 gas for all functions except for the constructor.

Chapter 6

Discussion

This discussion is divided into three parts, moving from the initial comparison of platforms to the proposition of which differences are meaningful. First, differences between platforms are determined and discussed. Second, the significance of these differences is evaluated through the experiences gained during development. The discussion reveals five differences between platforms, that were meaningful when developing a ROSCA application. The third and last section is devoted to a discussion about sustainability aspects considered in the context of the study.

6.1 Comparison of platforms

The comparison of platforms was done through analyzing the data presented about the different properties in chapter 4. The sectioning is a result of this analysis, rendering a list of distinct differences between the platforms.

6.1.1 Need of trust

A difference between consensus mechanisms in the platforms examined is their varying need of trust in individuals. In Hyperledger Fabric, one or more ordering nodes are appointed in the configuration of the network. This requires the ordering nodes to be trusted by all participants. In Neo, the validator nodes are elected through a majority decision among the holders of NEO. To be elected, a candidate needs to convince a majority of voters that it can be trusted. In Ethereum, the use of PoW enables a functioning blockchain without participants having to trust any specific individuals. The developer needs to consider, in Hyperledger Fabric and Neo, if the validators can be trusted. In the long run, this also applies to the intended user of the application. On Ethereum, trusting individual miners is not necessary, but rather in the functioning of the consensus algorithm and the Ethereum network.

6.1.2 Handling payments

Hyperledger lacks a pre-implemented currency or token for use in applications. An own token can be implemented, but that requires a thought over design and trust from the users. Neo has two globally tradeable currencies, NEO and GAS, which can be used for peer-to-peer payments. Unfortunately, neither one of them are compatible with being deposited into contracts. This complicates the development of any payment

application with the need for locking up value, and thereby an application specific token might need to be implemented. Ethereum has a well-integrated cryptocurrency, Ether, which is ideally suited for handling peer to peer payments. Since contracts are accounts, with the ability to hold Ether, escrow contracts can be implemented using the trusted, already established Ether as value unit. An escrow contract is a contract between two parties where funds for payments in the contract are deposited with a third party (Narayanan et al., 2016).

6.1.3 Permission and identification

In a Hyperledger channel, everyone included needs to be permissioned through the system configuration. This requires identification to some degree and implementation engagement of all parties. In Neo, only the validating nodes need to provide their real identity in contrast to users who can but don't have to. Important is that Neo is compatible with a widespread standard for digital identification, which can be useful if the application needs some level of identity. In Ethereum, are anonymous by default through the use of public keys. Identity can though be encoded in the smart contract scripts if needed.

6.1.4 Transaction throughput and speed

Out of the three platforms examined, Hyperledger Fabric has the highest transaction throughput of around 3000 TPS. Its consensus algorithm further gives the network immediate finality. Similarly, Neo has both a high transaction throughput, of 1000 TPS and the feature of immediate finality. Ethereum, in contrast, has a lower transaction throughput, as shown around 29 TPS. It also has probabilistic finality, giving longer times for being certain that transactions have been processed.

6.1.5 System stability

System stability can be discussed in terms of decentralization of the different networks. In a Hyperledger channel, one or a few nodes are handling the ordering, which makes it sensitive to downtime of orderers. Because of Neo's consensus model, becoming a validator node is a long process, including campaigning to gain trust within the network. This makes the validator role reachable for only a few serious actors. Contrasting is Ethereum, which consensus model theoretically lets anyone join into the validation process instantaneously. Worth mentioning is that even though this is supporting decentralization in theory, the actual degree of decentralization in Ethereum has shown to be questionable since a majority of miners join mining pools, as previously described.

Networks are at risk of being disrupted by different types of contingencies. In Hyperledger Fabric and Neo, a consensus is not achievable with more than $\frac{1}{3}$ of the nodes

dysfunctional or malicious. Ethereum and other PoW networks could suffer from 51 percent attacks if some malicious miner or mining pool gains control over too much computational power.

6.1.6 Development community

The current situation on Ethereum requires developers to learn Solidity. This can be compared with Neo, which is compatible with several generally adopted languages. However, both these platforms still require the development team to familiarize with the platforms' respective software development tools. A Hyperledger channel can be implemented according to developers' preferences, and chaincode is written in Go, node.js, or Java, which are all mainstream languages.

The community surrounding Ethereum is the most comprehensive one. This is reflected in both the number of contributors to the source code, but also in the extensive documentation and forums driven by community members. Hyperledger comes second, with a substantial set of source code contributors, but also seen to information and documentation. However, a lot of the material is provided by organizations such as IBM and Linux foundation, and there is not as much peer information to receive from other developers. Neo has a much smaller number of contributors, in both source code repository and forums. Most of the documentation is provided by the founding organization.

6.1.7 Cost of development

On the three platforms examined, there is a fundamental difference in how costs occur when launching an application. On Hyperledger, no fees are necessarily required for transacting and operating on the network. Instead, the costs connected to the application consists of such related to setting up and maintaining the infrastructure for establishing the network. On the contrary, transactions and system operations on Neo and Ethereum are paid for on chain, with the use of cryptocurrencies. For the individual developer however, neither time nor resources needs spending in establishing an infrastructure.

Even if the cost structures on Ethereum and Neo are similar in terms of handling payments, the cost levels varies. Deploying a contract is cheap on Ethereum, and in line with doing other transactions and operations on the system. On Neo, contract deployment is expensive, whereas other transactions and operations has lower or even non-existing fees. Furthermore, Neo can handle a vastly higher transaction throughput and therefore indicates lower congestion.

The high deployment cost on Neo is a barrier that the founding organization has decided to use, possibly to limit the amount of contracts being deployed. Developers can request gas from the organization to cover deployment costs. This leads to a sort of filtering, ideally raising the quality of deployed projects in the network.

6.2 Establishment of framework

In this section, differences are pointed out as meaningful or not based on the experience gained from developing. The relevant experience includes platform selection, development of the contract and obstacles encountered during the development process.

6.2.1 Meaningful differences

Based on what was presented in the development chapter, the following differences between platforms is to be considered as meaningful in the context of a ROSCA application.

Trust

As section 6.1.1 pointed out, Ethereum offers a consensus mechanism where the trust of the network is not connected to trust in any individual. People utilizing the network do not have to trust each miner, as long as the consensus algorithm and network of miners are deemed reliant. Because of this, not only the developer but also the users of the application, do not need to engage in assessing the trustworthiness of validators. In Neo and Hyperledger, the networks require trust in certain individuals.

Hiding data on a public blockchain

Hiding data was deemed necessary during the application development. Since the data stored on the blockchain is distributed, this is an inherent challenge. Without the possibility of hiding data, the viability of the contracts is greatly affected. If the content of the contract is accessible to everyone, then the contract is also exposed to malicious users taking advantage of improper implementation. A practical example of how the viability of a contract is affected negatively when data are left public is discussed in section 5.4.1. The section also illustrates how data can be hidden on a public blockchain. However, the possibility of doing this is dependent on the developers' capability of implementing thought over script solutions, which the section exemplifies. For the development, this was an obstacle that could have been easily solved on a permissioned blockchain such as Hyperledger Fabric, where the access can be restricted. One way of hiding data on a public blockchain such as Ethereum is to use cryptography, as shown in chapter 5.

Development community

The development community supporting Ethereum was of great importance to support the development of the application. This was showcased several times during the development process. For example, the solution using cryptography to hide data was

attained by the development community. The comprehensive material provided by the community such as documentation, forum, and development tools enabled the development team to construct a contract even though no previous knowledge existed. This allows developers to utilize the network with a lower entry barrier. During the initial platform examination and selection, it was found that Neos' community was not nearly as comprehensive as Ethereums'. Finding answers to questions on Neo was hard and developing under such conditions would have been aggravating to the development. Hyperledgers' community is strongly connected to large corporate organizations, and its content is therefore not as focused on helping individual developers.

Handling payments

For the development of an application that comprises value transfers, the way of handling payments on the network is necessary to analyze. This, because the payments need to be conducted as efficiently as possible. Ethereum enables seamless payments with the global cryptocurrency Ether. The developer can implement payments between accounts by simply calling a function on the EVM. This would not have been as easy on the other two platforms for reasons described in section 6.1. Since this project is focused on blockchain and the technology, off-chain transactions were not considered an alternative when developing the ROSCA. In another situation where the goal is to build the optimal application in terms of customer value versus costs, on-chain transactions may be an alternative. In such a case, the blockchain part of the application is only used for bookkeeping of transactions and agreements.

Cost of development

When developing an application on Ethereum the cost of deploying a contract is low in relation to that on Neo. This was an important decision criteria for selecting Ethereum for development, further motivated by the consideration of using multiple smart contracts for the application. It should be emphasized that deployment and transaction costs are the only blockchain specific costs that need to be considered on Ethereum and Neo. On Ethereum, these costs are transparent and were easily assessed with help of the Remix IDE debugger. Development on Hyperledger would not only have required higher implementation costs, but also a more complicated estimate of application hosting costs, since they are determined by the off-chain costs of running the network. In addition, there may be differences in development costs in terms of, for instance, salaries for developers and time required to develop applications

6.2.2 Insignificant differences

Not all differences between the platforms were perceived as decisive when developing a ROSCA application. These may though be more important when developing another application and therefore, they will be discussed briefly.

Transaction throughput and performance

The transaction throughput on the different platforms did not play a decisive role in the choice of platform to develop on. The reason for this is that the application only needs to have transactions processed around once a week, which all platforms handle without problems. Platforms such as Neo offers a better transaction throughput, which is of importance when developing a scalable application that requires more frequent transactions.

When developing applications on platforms such as Ethereum, having probabilistic finality, the users of an application can not be fully guaranteed that the agreed-upon transaction immediately ends up on the long term chain. As for any application including value transactions, this is negative for the application developed in this project, since it can result in varying transaction time.

System stability

When developing applications on platforms, the number of validating nodes currently in the system needs to be considered. This can be assumed to be especially important when developing an application intended to handle value transactions. In this aspect, developing a ROSCA on the Ethereum network is a logical choice since the network has a large amount of validating nodes and it is the only one of the three examined networks that utilize an objectively verifiable consensus algorithm which can be assumed to be of strong importance to risk-averse users. A potential downside with developing on the Ethereum network is the networks intention to change from a proof-of-work consensus algorithm to one utilizing a proof-of-stake algorithm. This adds uncertainty in how the networks performance and stability will be affected by such a change.

Permission and identification

In theory, Ethereum applications can be built without the identification of users. In our ROSCA implementation, students decide to participate together with people they know and therefore, can identify each other with the use of public keys and additional identification parameters described in the development chapter. However, the students cannot exclude anyone from the contract, which is possible in Hyperledger. If the application were to enable people not knowing each other to be part of the same circle, some identity or social collateral would be required. Compared to Neo, which is compatible with Public Key Infrastructure, developing on Ethereum requires a bit more creativity and work regarding these questions. One example is to force users to identify themselves by social security number or some other form of identity marker.

6.2.3 Areas of usage

Hyperledger Fabric is developed mainly for business-to-business use cases. As already discussed, it requires a more extensive commitment by developers and users in terms of development costs and physical resources. Platforms such as Neo and Ethereum are generally better suited for applications that are of traditionally business-to-customer character, due to the low need for resource commitment from users. This is necessary for gaining adoption by students targeted as users of the application since they are typically not willing to engage in maintaining the network.

The suitability of developing the ROSCA application on Ethereum was obvious, not at least because of the supply of examples of other similar solutions. The development process was, therefore, in a large extent, a recombinatorial process. In general, Hyperledger is most suitable for blockchain applications involving corporations or bigger organizations. All kinds of business-to-business solutions where parties trust in each other enough to elect ordering nodes are good fits for Hyperledger since it is often computationally more effective. As mentioned earlier in the report, one of the selling arguments for the Neo platform is that it is enabling a smart economy. This is done by combining smart contracts, digital identity, and digitized assets. This makes tracking ownership of physical assets an obvious use case for the Neo platform.

6.3 Sustainability

There are many questions regarding sustainability that could be discussed concerning this project. This section aims to deal with the issues that have arisen and been considered most relevant during the project. One popular way to decompose the wide concept of sustainability is to divide the overarching term into ecological, economic, and social sustainability. In this section, social and economic sustainability is merged into one subsection since these overlap to a great extent.

6.3.1 Ecological sustainability

A commonly discussed topic relating to blockchain is the energy consumption of networks using PoW. As known, the mining on those networks is a race to finding the next block, with the probability of winning being reliant on the amount of computational power possessed by each miner. Since the start of the well known Bitcoin network, miners have gone from performing mining on personal computers to requiring application specific hardware set up by organizations in large server halls (Narayanan et al., 2016). The Ethereum network follows the same pattern (Buterin, 2019). Since the difficulty of finding new blocks is adjusted to retain the same block time as the total amount of computational power is increasing, more and more energy is consumed to power the network. An estimation of the energy required to run the Bitcoin network from 2015 revealed that it consumed around 10 percent of the energy produced by an

average nuclear plant (Narayanan et al., 2016). As the network expands, the energy consumption follows, and some people claim that Bitcoin is as of spring 2019 using more energy than entire countries such as Austria (Urwin, 2018).

It should be noted that Bitcoin, as well as Ethereum, today only processes minor fractions of the transaction amount being handled by for example the Visa platform (Buterin, 2019). Scaling up a PoW network to such dimensions would seriously increase its energy consumption. This issue is one of the reasons networks with alternative consensus mechanisms, such as Neo and Hyperledger Fabric, are gaining popularity. To do a proper evaluation of the energy consumption, this must be compared to the alternatives which may, for instance, be minting coins and printing banknotes or energy used by other parts of the current financial system.

6.3.2 Social and economic sustainability

Social sustainability means that society is characterized by equity and equality and that people live healthily and with trust and confidence in each other. Economic sustainability is more diverse when it comes to the definition. Here, economic sustainability is defined as long term sustainable householding of material and human resources.

One significant economic problem with blockchain platforms is the volatility in the cryptocurrencies many of them rely on, which historically has been high and seemingly unpredictable (CoinMarketCap, 2019). A potential scenario is where a financial application of some form uses cryptocurrency as a complement to fiat currency. Fiat currency refers to money that lacks backing from a physical commodity such as gold or real estate and instead is issued with the government as the mechanism ensuring the value (Chen, James, 2019). Stablecoins has been presented as one possible solution to this. A Stablecoin is a cryptocurrency backed by some other kind of asset or collection of assets (Wikipedia, 2012). Bilal Memon presents four types of stablecoins. Three backed by assets and one that is more like fiat currency. The three asset-backed are backed by a physical commodity, fiat currency, or another cryptocurrency such as Bitcoin or Ether. The last one is called seignorage-style stablecoin and is a kind of stablecoin that is supposed to work like a fiat currency but with algorithms governing the money supply and price levels instead of a central bank (Memon, 2018). This becomes an important question when a value is starting to be stored on a blockchain and measured in units of cryptocurrency. Note here that these four categories are not necessarily exhaustive. One may imagine other kinds of stablecoins backed by, for instance, a government instead of an algorithm.

A functioning credit system is key for the economy (*Finansinspektionen*, 2019). Today the credit system is regulated and primarily controlled by the banks and by the state/-government. A system like ROSCA is different in that aspect. A ROSCA with pure P2P-lending will not rely on some third party, ensuring trust and reliability. There will neither be any central point of risk management. Uncontrolled and excessive credit growth is a substantial risk to an economically sustainable society. There is research proposing that the risks of eliminating the banks and letting people save and lend peer

to peer could even lower the risks of default (Chami & Fischer, 1995). One important thing to mention is that ROSCAs, as they exist today, is mainly among people closely related to each other. The new possibilities that a blockchain based ROSCA would generate means that more weakly connected people could co-participate in the same ROSCA. This means that the effects with social mechanisms lowering the default rate may diminish. A problem with such reasoning is the lack of research in this area, which means that it is highly speculative and outside the scope of this study.

One actor or a group of actors controlling a credit system may impose a risk to a socially sustainable society if that means that some people are excluded from the financial system. The reason for this exclusion may, in many cases, be well substantiated from the banks' point of view since they have to control their risks in some way. A ROSCA and P2P-finance, in general, is eliminating this intermediary, and the system's foundation is that it automatically balances saving and lending.

The anonymity and pseudonymity that many blockchain platforms offer can also be a problem in relation to the legal system and the ability to maintain the existing tax system when the government cannot track individuals' transaction history and assets or, for instance, force banks to freeze accounts.

Chapter 7

Conclusion

Throughout the process of this study, the versatility of blockchain platforms has become prominent. The modularity of blockchain technology results in a diversity of platform behaviour and properties. The lack of tools that can be used to compare blockchain platforms motivates the purpose of this study, to propose a framework that compares different blockchain platforms, and also to evaluate the framework through the development of a ROSCA application.

Properties for the framework were selected through a literature review and a platform examination of three operating blockchain platforms. The three platforms were selected using Zamfir's triangle with the intent to represent a diversity of blockchain platforms and thus form a comprehensive foundation for later assessment. Properties were categorized under three subsections, system design, system performance, and platform ecosystem. Further details about the properties can be found in Appendix A.

Five properties of the platforms showed to be meaningful for the process of developing a ROSCA application on one selected platform, Ethereum. These five are composed in Table 7.1. First, Ethereum's trustless consensus model did not require establishment of trust in anyone specific, neither from the developers' nor users' perspective. This eliminates the need for an outside or inside controller of the ROSCA application. Second, the level of privacy and access on Ethereum lead to a programmatic challenge in ensuring a secure contract. Third, the handling of payments was highly facilitated through the seamless payments with Ethereum's own cryptocurrency Ether. Fourth, the transparent on-chain transactions and execution costs on Ethereum made the cost base easy to assess. Fifth and last, the Ethereum development community was heavily utilized throughout the entire process, and provided the developers with the majority of the information needed.

Platform comparison framework	
Property	*Description*
Trust	Trust in individuals or only in an algorithm
Privacy and access	Permissioned or public
Handling payments	On chain cryptocurrencies and tokens or off chain transactions
Cost of development	On chain transaction and execution costs or off chain costs for maintaining network
Developer community	Number of contributors to system, forum strength and development tools available

Table 7.1: Suggested meaningful properties for development of a ROSCA application on a blockchain platform, identified after development of a ROSCA application on Ethereum.

7.1 Directions for future research

This study forms a foundation for further research about frameworks for comparing blockchain platforms. The identified properties of blockchain platforms can be separately evaluated and assessed. The selection of properties can also be compared as a framework against other frameworks, including Zamfir's triangle, to better determine advantages and disadvantages.

To further strengthen the framework, additional development projects is recommended. A variety of developed applications would give insights from blockchain platforms in different situations and usage areas. Also, continued work with the development of the ROSCA application might generate further insights about blockchain platform properties, in a later stage of the application's life cycle. Ideally, a systematic and generalized framework would exist to enable developers and other stakeholders to compare blockchain platforms with regard to the design requirements for specific applications.

References

Altmann, P. (2019). *Consensus Mechanism: A brief introduction.*

Ambec, S., & Treich, N. (2007). Roscas as financial agreements to cope with self-control problems. *Journal of development economics*, *82*(1), 120–137.

Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., ... others (2018). Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth eurosys conference* (p. 30).

Bach, L., Mihaljevic, B., & Zagar, M. (2018). Comparative analysis of blockchain consensus algorithms. In *41st international convention on information and communication technology, electronics and microelectronics (mipro)* (pp. 1545–1550).

Blomkvist, P., & Hallin, A. (2015). *Method for engineering students - Degree projects using the 4-phase model*. Lund: Studentlitteratur AB.

Buterin, V. (2019). *Ethereum White Paper: A next-generation smart contract and decentralized application platform.* Retrieved 2019-04-26, from `https://github.com/ethereum/wiki/wiki/White-Paper`

Castro, M., & Liskov, B. (1999). Practical byzantine fault tolerance. In *Osdi* (Vol. 99, pp. 173–186).

Chami, R., & Fischer, J. H. (1995). Community banking, monitoring, and the clinton plan. *Cato Journal*, *14*, 493.

Chen, James. (2019, 4). Fiat Money. *Investopedia*. Retrieved 2019-05-14, from `https://www.investopedia.com/terms/f/fiatmoney.asp`

CoinMarketCap. (2019). *Top 100 Cryptocurrencies by Market Capitalization.* Retrieved 2019-05-14, from `https://coinmarketcap.com/`

David Göthberg. (2019). *Merkle Tree.* Retrieved from `https://en.wikipedia.org/wiki/Merkle_tree`

Ethereum. (2019). [Source code repository]. Retrieved 2019-05-14, from `https://github.com/ethereum`

Ethereum. (2019, 4 30). *Solidity Documentation v0.5.8.* Retrieved 2019-05-14, from `https://buildmedia.readthedocs.org/media/pdf/solidity/v0.5.8/solidity.pdf`

Etherscan. (2019). *Ethereum (ETH) Blockchain Explorer.* Retrieved 2019-05-14, from `https://etherscan.io/`

Finansinspektionen. (2019). Retrieved 2019-05-14, from `https://www.fi.se/`

Flick, U. (2014). *An introduction to qualitative research* (5th ed.). Sage Publications Limited.

Gencer, A. E., Basu, S., Eyal, I., Van Renesse, R., & Sirer, E. G. (2018). Decentralization in bitcoin and ethereum networks. *arXiv preprint arXiv:1801.03998*.

Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., & Gervais, A. (2019). *Sok: Off the chain transactions.* Retrieved from https://eprint.iacr.org/2019/360

Holzer, A., & Ondrus, J. (2011, 2). Mobile application market: A developer's perspective. *Telematics and Informatics*, *28*(1), 22–31.

Hyperledger. (2018, 6). *An Introduction to Hyperledger.* Retrieved from https://www.hyperledger.org/wp-content/uploads/2018/07/HL_Whitepaper_IntroductiontoHyperledger.pdf

Hyperledger. (2019). *Hyperledger fabric.* [Source code repository]. Retrieved 2019-05-04, from https://github.com/hyperledger/fabric

Hyperledger. (2019a, 5 13). *Hyperledger Fabric Documentation Release 1.4.* Retrieved 2019-04-26, from https://buildmedia.readthedocs.org/media/pdf/hyperledger-fabric/release-1.4/hyperledger-fabric.pdf

Hyperledger. (2019b). *Hyperledger - Open Source Technologies.* Retrieved from https://www.hyperledger.org/

IBM. (2019). *What is blockchain?* Retrieved 2019-05-07, from https://www.ibm.com/blockchain/what-is-blockchain

Infura. (2019). *Scalable Blockchain Infrastructure.* Retrieved 2019-05-14, from https://infura.io/

J. Besley, T., Coate, S., & Loury, G. (1993, 01). The economics of rotating savings and credit associations. *American Economic Review*, *83*, 792-810.

Jeffries, A. (2018). 'Blockchain' is meaningless. *The Verge*. Retrieved 2019-04-26, from https://www.theverge.com/2018/3/7/17091766/blockchain-bitcoin-ethereum-cryptocurrency-meaning

Kabuya, F. I. (2015). The rotating savings and credit associations (roscas): Unregistered sources 0f credit in local communities. *Journal Of Humanities And Social Science*, *20*(8), 95–98.

Kedir, A. (2015, 05). The economics of rotating savings and credit association: Evidence from ethiopia. In *Third international conference on development studies in ethiopia.*

Lamport, L., Shostak, R., & Pease, M. (2002, 10). The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, *4*(3), 382–401.

Mbizi, R., & Gwangwava, E. (2013, 01). Rotating savings and credit associations: An alternative funding for sustainable micro enterprise-case of chinhoyi, zimbabwe. *Journal of Sustainable Development in Africa*, *15*.

Memon, B. (2018). Guide to Stablecoin: Types of Stablecoins & Its Importance. *Master The Crypto*. Retrieved 2019-05-14, from https://masterthecrypto.com/guide-to-stablecoin-types-of-stablecoins/

Narayanan, A., Bonneau, J., Felten, E., Miller, A., Goldfeder, S., & Clark, J. (2016). *Bitcoin and Cryptocurrency Technologies Introduction to the book*. Princeton University Press.

NEO Documentation. (2019). Retrieved 2019-05-14, from https://docs.neo.org/en-us/index.html

Oxford dictionary: Blockchain. (2010, 01). Retrieved from https://

www.oxfordreference.com/view/10.1093/acref/9780199571123.001.0001/m_en_gb1003287

Panetta, K. (2017). Top Trends in the Gartner Hype Cycle for Emerging Technologies, 2017. *Smarter with Gartner*. Retrieved 2019-04-26, from https://www.gartner.com/smarterwithgartner/top-trends-in-the-gartner-hype-cycle-for-emerging-technologies-2017/

Panetta, K. (2018). 5 Trends Emerge in the Gartner Hype Cycle for Emerging Technologies, 2018. *Smarter with Gartner*. Retrieved 2019-04-26, from https://www.gartner.com/smarterwithgartner/5-trends-emerge-in-gartner-hype-cycle-for-emerging-technologies-2018/

Project, N. (2019). *Neo smart economy.* [Source code repository]. Retrieved 2019-05-14, from https://github.com/neo-project/neo

PwC. (2016, 2 22). *Making sense of bitcoin, cryptocurrency and blockchain.* Retrieved 2019-05-14, from https://www.pwc.com/us/en/industries/financial-services/fintech/bitcoin-blockchain-cryptocurrency.html

Satoshi Nakamoto. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System.* Retrieved from https://bitcoin.org/bitcoin.pdf

Shneiderman, B. (2016). *The new ABCs of research - Achieving Breakthrough Collaborations*. Oxford: Oxford University Press.

Tuan Anh Dinh, T., Wang, J., Chen, G., Liu, R., Chin Ooi, B., & Tan, K.-L. (2017, 05). Blockbench: A framework for analyzing private blockchains. In (p. 1085-1100).

Urwin, R. (2018, 8). Mining for bitcoin 'uses more energy than Austria', says PwC's Alex de Vries. *The Sunday Times*. Retrieved 2019-05-14, from https://www.thetimes.co.uk/article/mining-for-bitcoin-uses-more-energy-than-austria-says-pwcs-alex-de-vries-thnvjq55b

van Steen, M., & Tanenbaum, A. S. (2018). *Distributed Systems* (3rd ed.).

Vitalik Buterin. (2014). *Proof of Stake: How I Learned to Love Weak Subjectivity.* Retrieved 2019-05-14, from https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity/

Vrba, T. (2018, 4). *Decentralized governance, reputation and mechanism design for a global social impact funding platform on a public blockchain.*

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

Wang, H., Zheng, Z., Xie, S., Dai, H. N., & Chen, X. (2018, 10). Blockchain challenges and opportunities: a survey. *International Journal of Web and Grid Services*, *14*(4), 352.

Wikipedia. (2012). *Stablecoin.* Retrieved from https://en.wikipedia.org/wiki/Stablecoin

Wood, G. (2014). *Ethereum: A Secure Decentralised Generalised Transaction Ledger.* Retrieved 2019-05-14, from https://ethereum.github.io/yellowpaper/paper.pdf

Wright, A., & De Filippi, P. (2015). Decentralized blockchain technology and the rise of lex cryptographia. *Available at SSRN 2580664*.

Yli-Huumo, J., Ko, D., Choi, S., Park, S., & Smolander, K. (2016). Where is current

research on blockchain technology?—a systematic review. *PloS one*, *11*(10), e0163477.

Zamfir, V. V. (2017). *I think this is a fundamental tradeoff in fault tolerant consensus protocols. (Thread!).* [Tweet]. Retrieved from `https://twitter.com/vladzamfir/status/942271978798534657`

Zervas, G., Proserpio, D., & Byers, J. W. (2017). The rise of the sharing economy: Estimating the impact of airbnb on the hotel industry. *Journal of marketing research*, *54*(5), 687–705.

Appendix A

Platform examination questions

Platform examination framework	
System design	
Data structures	What data structures are used by the platform? Is the state of the blockchain stored separately?
Participants	What nodes participate on the platform? What functions do the different nodes have?
Consensus mechanism	What consensus mechanism is used on the platform? How is the consensus mechanism implemented, considering consensus model and consensus algorithm? In what way is the incentive structure for participants designed?
Value on the system	Is there ant inherent cryptocurrency on the platform? How can this cryptocurrency be used?
Interacting with the system	How does the platform allow for outside interactions? What transactions are possible on the platform?
Access and privacy	When and how do different participants need to be identified? How do users access the platform? What can be said about the platform's privacy?
System performance	
Computing performance	How many transactions per second can be done on the platform? How was the transaction speed tested or calculated? Does blockchain enable immediate finality or do participants need to wait to ensure transactions are valid?
System stability	What can be said about the platform's stability? Are there any possible platform attacks?
Platform ecosystem	
Programming possibilities	What programming languages are supported for development on the platform? How can the platform be customized for the purposes of the application?
Developer community	How wide is the community around the platform? Is any source code available? Are there open source tools or standards for interacting with the platform?
User Access	How accessible is the platform for the potential usebase of the application?
Hosting costs	What is the transaction cost on the platform? What is the cost for contract deployment? Are there any other costs connected to the platform?
Areas of usage	Which are the actual and intended areas of usage for the platform?

Table A.1: This table describes the areas used to describe and compare different blockchain platforms. All areas have questions that further specify what information is relevant for each area.

Appendix B

ROSCA implementation

```
pragma solidity >=0.4.22 <0.6.0;

contract circle{

    uint circleAmount;
    uint circleInterval;
    uint circleStartdate;

    struct Participant {
        string name;
        string cid;
        string division;
        bool gotPaid;
        bool hasPaid;
        uint bid;
    }

    mapping (address => Participant) participants;
    address payable[] public participantsAddresses;

  constructor(uint amount, uint interval, uint startDate) public{
      circleAmount = amount;
      circleInterval = interval;
      circleStartDate = startDate+now;
  }

  function pay() public payable{
      require(isInCircle(msg.sender)==true,"you are not part of the
           circle");
      require(msg.value == circleAmount, "you have not payed the
          specified amount");
      participants[msg.sender].hasPaid=true;
  }

  function payOut() public {
      require(circleStartdate<now);
      (address payable x, uint y) = getHighestBidder();
      x.send((circleAmount-y)*(participantsAddresses.length-1)+
          circleAmount);
      participants[x].gotPaid=true;
      participants[x].hasPaid=false;
      participants[x].bid=0;
      for(uint i=0; i<participantsAddresses.length-1; i++){
          if(x!=participantsAddresses[i]){
              participantsAddresses[i].send(y/(
                  participantsAddresses.length-1));
              participants[participantsAddresses[i]].bid=0;
              participants[participantsAddresses[i]].hasPaid=false;
```

```
45
46              }
47          }
48        circleStartDate += circleInterval;
49      }
50
51      function joinCircle(address payable _address, string memory
            _name, string memory _cid, string memory _division) public
            {
52          participants[_address] = Participant({
53              name:_name,
54              cid:_cid,
55              division:_division,
56              gotPaid:false,
57              hasPaid:false,
58              bid:0
59          });
60          participantsAddresses.push(_address);
61      }
62
63      function placeBid(int myBid) public{
64          require(isInCircle(msg.sender)==true,"you are not part of
                the circle");
65          participants[msg.sender].bid = uint(myBid);
66      }
67
68      function getHighestBidder() view private returns(address
            payable, uint){
69          uint higest=0;
70          uint secondHighest=0;
71          address payable tempAddress;
72          for(uint i=0; i<participantsAddresses.length; i++){
73              if(participants[participantsAddresses[i]].bid>=higest){
74                  secondHighest = higest;
75                  higest = participants[participantsAddresses[i]].bid
                        ;
76                  tempAddress = participantsAddresses[i];
77              }else if(participants[participantsAddresses[i]].bid>=
                    secondHighest){
78                  higest=secondHighest;
79              }
80          }
81          return (tempAddress, secondHighest);
82      }
83
84      function isInCircle(address _address) private returns(bool){
85        for(uint i = 0; i<participantsAddresses.length; i++){
86            if(_address==participantsAddresses[i]){
87                return true;
88            }
89        }
90        return false;
91    }
92
93  }
```

Appendix C

Blind auction implementation

```
contract BlindAuction{

    struct participant{
         bytes32 hiddenBid;
    }

    address highestBidder;
    uint highestBid;

    uint Biddingtime;
    mapping(address => participant) participants;
    address[] participantsAddresses;

    constructor(uint _Biddingtime) public{
        Biddingtime = _Biddingtime;
    }

    function addParticipant(address adr) public{
        participantsAddresses.push(adr);
        participants[adr] = participant(0);
    }

    function placeBid(bytes32 secretBid)public{
        require(now<Biddingtime);
        participants[msg.sender].hiddenBid = secretBid;
    }

    function reveal(uint bid) public returns(bool){
        require(now>Biddingtime);
        if (keccak256(toBytes(bid))== participants[msg.sender].
            hiddenBid && bid>highestBid){
            highestBidder = msg.sender;
            highestBid = bid;
        }
    }

    function toBytes(uint256 x) private returns (bytes memory b) {
        b = new bytes(32);
        assembly { mstore(add(b, 32), x) }
    }
}
```