Henrik H. Rasmussen Kasper W. Mortensen

Networks and Distributed Systems
Master Thesis - 2018

Design, Development and Evaluation of BIG IoT Functionalities on the Ethereum Blockchain

Institute of Electronic Systems
Fredrik Bajers Vej 7
DK-9220 Aalborg Ø

AALBORG UNIVERSITET
STUDENTERRAPPORT

Title:
Design, Development and Evaluation of BIG IoT Functionalities on the Ethereum Blockchain

Theme:
Networks and Distributed Systems

Project Period:
Master Thesis

Project Group:
18gr1021

Participant(s):
Henrik Heegaard Rasmussen
Kasper Wissing Mortensen

Supervisor(s):
Hans Peter Schwefel
Lars Møller Mikkelsen
Tatiana Kozlova Madsen

Copies: 2

Page Numbers: 87

Date of Completion:
June 7, 2018

Abstract:

This report investigates the possibilities of combining the BIG IoT research project with the new technological concept of blockchain. In order to do so, both the BIG IoT system and the blockchain technology was analyzed in order to determine where the blockchain technology could provide a possible advantage. On the basis of this analysis several usecases has been conceptualized as a way of determining the usability of the blockchain technology. As blockchain is a relatively new technology, the state of art is more or less absent, and therefore this project has no project to lean on, and it was therefore needed to create a architecture that both supported blockchain and BIG IoT elements. All the design and implementation of the BIG IoT usecases was then build on the new architecture and with the implementation it was possible to gain a deeper understanding of the impact Ethereum has on the system. In order to make the design as robust as possible, an evaluation of the general known problems was created together with a theoretical model of inconsistency in the blockchan technology. This evaluation reveal several aspect of secure coding of smart contract, and where the new architecture has its limitation. As this project was time restricted, some areas was not covered and therefore further evaluations area of things as security, mining, Ethers and the inconsistency model is still needed as future work.

Preface

This reports documents the work done for the 10th semester Master Thesis in Networks and Distributed Systems at Aalborg University. The topic of this project is Design, Development and Evaluation of BIG IoT Functionalities on the Ethereum Blockchain. The project started 01/02/2018 and ended 07/06/2018. This project was done in cooperation with Aalborg University and the BIG IoT research project and in this context the authors would like to thank Tatiana Kozlova Madsen, Lars Møller Mikkelsen and Hans Peter Schwefel from Aalborg University for contextual and academic supervision throughout the master thesis.

This report is split into four parts. Part I contains an introduction to the problem as well as an understanding of the architecture of both the BIG IoT System, Ethereum Blockchain System and how to merge these two. Part II contains the implementation of both the testbed and the usecases. Part III contains performance analysis and evaluation. Part IV contains conclusions and future work. The chapters are enumerated as 1, sections as 1.1 and subsections as 1.1.1. Figures and tables will be enumerated in the order they appear in. The source references will be indicated as [1] and can be found in the back of the report. A list of enclosed files can be found in Appendix B.

Henrik Heegaard Rasmussen

Kasper Wissing Mortensen

Glossary

ABI Application Binary Interface

AMR All Matching Results

API Application Programming Interface

BIG IoT Bridging the Interoperability Gap of the IoT

CAP CAP theorem, also known as Brewer's theorem

CC Consumer Clients

CDF Cumulative Density Function

cpp-ethereum C++ for Ethereum

DAPP Decentralized Applications

EIP Ethereum Improvement Proposal

EnodeID Ethereum Node Identifier

EP Endpoint

EXP Exponential

FIFO First-In-First-Out Queueing scheme

EVM Ethereum Virtual Machine

FMR First Matching Result

GB GigaByte

Geth Go for Ethereum

Gwei 10^{-9} Ethers

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

ID Identifier

I/O Inputs and Outputs

IoT Internet of Things

IP Internet Protocol

IPC Inter-Process Communication

JSON JavaScript Object Notation

MA Moving Average

OID Offering ID

P2P Peer-to-Peer

PDF Probability Density Function

PHP Hypertext Preprocessor

PKI Public Key Infrastructure

POW Proof Of Work

Pyethapp Python for Ethereum

RFMR Random First Matching Result

RPC Remote Procedure Call

RTT Round-Trip Time

TxPool Transmission Pool of Ethereum nodes

UE User Equipment

URL Uniform Resource Locator

Contents

I Introduction **9**

1 Internet of Things **10**
1.1 BIG IoT . . . 11
1.2 BIG IoT Marketplace . . . 11

2 Blockchain **13**
2.1 Distributed Storage . . . 13
2.2 Blockchain Elements . . . 14
2.3 The Ethereum Blockchain . . . 15
2.4 Initial Problem Formulation . . . 20

3 Ethereum in BIG IoT **21**
3.1 Combined Architecture . . . 21
3.2 Access Management Usecase . . . 23
3.3 Exchange Usecases . . . 24
3.4 Summary . . . 28

4 Problem formulation **29**
4.1 Problem statement . . . 29

II Implementation **30**

5 Implementation Platform **31**
5.1 Geth . . . 31
5.2 Peer Discovery . . . 33
5.3 Truffle . . . 34
5.4 Testbed . . . 35

6 Offering Usecase Implementation **38**
6.1 Create offerings . . . 38
6.2 Delete offerings . . . 40
6.3 Modify offerings . . . 41
6.4 Test of offering implementation . . . 41

7 Offering Query Usecase Implementation **43**
7.1 Test of offering query implementation . . . 44

8 Subscription Usecase Implementation **46**
8.1 Creating subscriptions . . . 46
8.2 Deleting subscriptions . . . 47
8.3 Test of subscription implementation . . . 47

9 Access Management Usecase **48**
9.1 Chain access on foreign nodes . . . 48
9.2 Test of Access Management Usecase . . . 49

III Evaluation **51**

10 Evaluation Areas **52**
10.1 Time Aspects . . . 52
10.2 Security . . . 52
10.3 Inconsistency . . . 53
10.4 Availability . . . 53

11 Time Aspects **54**
11.1 Offering Usecase . . . 54
11.2 Offering Query Usecase . . . 59

12 Security Concerns **65**
12.1 General Blockchain Security Concerns . . . 65
12.2 General Project Security Concerns . . . 67

13 Inconsistency **68**
13.1 Forking Problem . . . 68
13.2 Forking Probability Model . . . 69
13.3 Simulation . . . 72
13.4 Summary . . . 75

14 Availability **77**
14.1 Example Availability Calculation . . . 78

IV Final Remarks **79**

15 Conclusion **80**

16 Future Work **81**
16.1 Ethers and Mining . . . 81
16.2 Solidity Optimization . . . 82
16.3 Availability Optimization . . . 82
16.4 Emulation Of Forking Model . . . 82

Bibliography **83**

A Gas fee table [12] **86**

B Enclosed Files **87**

Part I

Introduction

Chapter 1

Internet of Things

Internet of Things (IoT) is a network of is the concept of different types of devices, such as sensors, vehicles, embedded hardware and home appliances, which are interconnected allowing them to communicate. IoT is growing rapidly and according to Intel it is expected that the amount of IoT devices will increase from 15 billion in 2015 to around 200 billion 2020 [7]. One of the reasons behind this rapid growth can be found in the fact that IoT devices can be combined to create complex features. This is because one of the primary features of IoT devices is the ability to exchange data, thereby allowing developers to build advanced applications, which take in a large variation of data types to perform actions. An example of an application is smart parking, which not only shows available parking lots, but can also calculate the fastest route to an optimal parking lot, by using information such as traffic density, price of parking and parking time limits. This data could be collected using IoT sensors.

One problem with IoT is that either the developer would have to own all the devices themselves or they would have to gather the data from other IoT platforms. Creating and running an IoT network is expensive and this makes it difficult for smaller companies to enter the IoT market. The high maintenance cost makes purchasing the data from a third party more of an appealing idea. One of the challenges here is that developers tend to create their own platforms, which might be incompatible with each other. A developer would have to discover who has the kind of data of interest and thereafter have to deal with the issues of potential incompatibility between the different data sources. This makes it difficult to create an application as described above. This is illustrated on Figure 1.1 and to distinguish between the roles of supplying data and utilizing the data, users who make data available will be called Providers and users who access/uses data will be called Consumers. Every interaction between the Consumer and the different IoT platforms goes through unique APIs because each Provider has implemented their own unique system and structure. To distinguish between unique APIs different geometric shapes are used. As seen on the figure, it can quickly

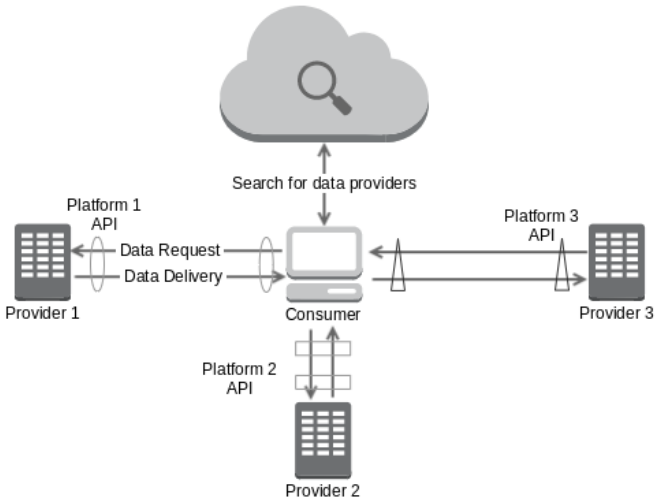


Figure 1.1: Illustration of the general structure of an IoT data sharing setup.

grow out of proportion. This is one of the major challenges in the current IoT landscape as it makes it difficult to utilize available information. This is the motivation behind the BIG IoT research project which

focus is to make IoT data more accessible.

1.1 BIG IoT

Bridging the Interoperability Gap of the Internet of Things (BIG IoT), is a project between multiple companies and universities around Europe and the project is funded from EU's Horizon 2020 [28]. The first goal of BIG IoT is to create a common API for data sharing and access. The common APIs are illustrated as ellipses in Figure 1.2 and makes it much simpler for a customer to access the purchased data as they do not have to develop customized access modules for each company. The second part of the BIG IoT platform is a marketplace. The general idea is to create a marketplace where companies can sell their data and interested buyers can purchase it. By collecting this onto a single location it makes the discovery and sharing of data between Provider and Consumer easier. This also allows Providers to monetize their IoT assets, thereby allowing them to profit from utilizing the BIG IoT platform. Figure 1.2 is a very general description of

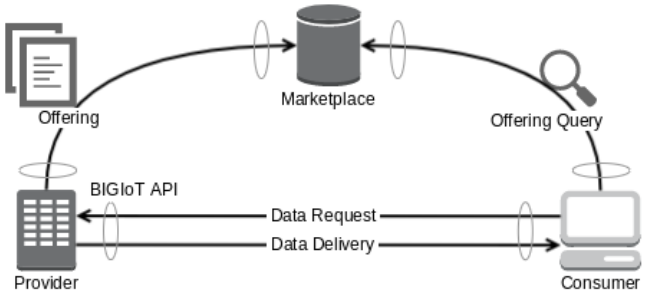


Figure 1.2: An illustration of the general structure of the BIG IoT setup.

how the system works, but it represents the relationships between different components in the system. To discover each other both Provider and Consumer has access to the marketplace.

1.2 BIG IoT Marketplace

The marketplace is where data is put up for sale. The marketplace is structured as seen on Figure 1.3. The marketplace contains all the functionality required to make agreements between Provider and Consumer. Any user who desires to utilize the system must create an organization. This organization is used to manage

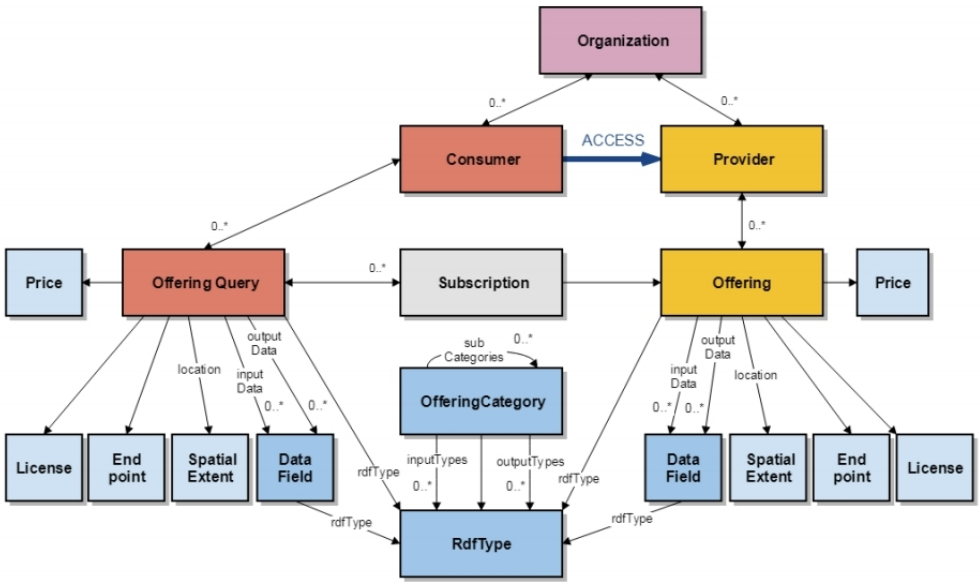


Figure 1.3: A diagram of the general structure of the BIG IoT marketplace. Illustration is from [3].

potential Providers and Consumers and allows a company to both provide certain data while consuming others. The Provider role can create listings for data called offerings. When a Provider creates a listing in the marketplace, its offering becomes visible to all Consumers. This offering contains all the relevant information regarding the data in question, such as:

- **Data field:** Type of data (weather, pollution, density, etc)
- **Spacial extent:** Physical location of data measurements
- **License:** License agreement for access of data (free, per month payment, one time payment, per access payment, per byte accessed)
- **Price:** The price of an offering depends on the license
- **Endpoint:** Access address of the actual IoT data

This information is then used when a Consumer searches the marketplace for suitable data. The Consumer performs an offering query, which is a search in the marketplace based on a given set of parameters. Once the Consumer has found the correct offering they can subscribe to it and access the data via the API. To achieve this functionality the marketplace contains several different functionalities. The system consists of three main areas of functionality with two more planed:

Access Management: The login system of the marketplace and the functionality which controls how to get access to the marketplace. The marketplace currently uses the OAuth2 protocol[3] for identity authentication which is used to verify if a user.

Exchange: The Exchange manages the sharing of data where the Providers can create and manage offerings and Consumers can search through available offerings. This means any offering created by the Provider is managed through this as well as any search and purchase made by the Consumers.

Accounting: Tracks the usage of data by a Consumer. This is necessary so any eventual payment based on data usage can be performed. This also allows further analysis of the activity of data based on regions, types, etc.

Charging: (*Not yet implemented*) Takes the accounting information and combines it with the agreed upon price for data to calculate the total sum of payment.

Billing: (*Not yet implemented*) A billing system which uses the accounting and charging modules to create bills for Consumers.

These areas represent the entirety of the BIG IoT marketplace in its current format. Because the BIG IoT platform is a research project there is always an interest in looking into new ways and technologies which can improve the system. One technology which has garnered a lot of attention over the last year is blockchain. Blockchain has primarily been used for crypto-currencies but its potential in areas such as security has drawn hopes of other implementation areas. BIG IoT therefore wants to investigates how blockchain technology can be used inside the system. The goal of this project is therefore to conceptually and experimentally evaluate the benefits and trade-offs of using the blockchain inside the BIG IoT marketplace.

Chapter 2

Blockchain

Blockchain is the common name for a continuous chain of blocks containing records stored in a distributed manner. The blocks are closely linked together with the cryptography method of hashing, where each block contains the hash of the previous block. All the members in the network will help share the newest version of the chain. The origin of blockchain is unknown, but the first examples of a secure chain was published in 1991, which also contained the aspect of distributed trust [14]. The first appearance of blockchain as known today, was created by the unknown author(s) Satoshi Nakamoto in 2008, with the creation of the crypto-currency Bitcoin [22]. As Blockchains stores the data in a distributed manner, it can be described as a distributed storage technology.

2.1 Distributed Storage

Distributed storage is a method of having data located at multiple locations and can be split into two categories:

- **Replication:** Replication uses mechanisms to determine what changes has been made in the storage and adopting these changes. This ensures that all nodes have the same image of the data. A change made in a node is then adopted by all other nodes. [21]
- **Duplication:** Performs much the same task a replication with the core difference that in duplication every node gets data from a master node. The master node is seen as the correct image of the data and all other nodes contain duplicate data versions of this one master node. Duplication is essentially a remote backup of data as seen in data clusters.

Blockchain uses replication as every node in the network helps share changes in the chain, which in blockchain means that every node helps distribute new blocks. This means that every node in the network contains the same copy of the data. This leads to two issues: large data consumptions and consistency in the data.

2.1.1 Blockchain Chain Size

How fast a chain will grow in size depends on the implementation and will depend on two factors: How fast new blocks are generated and how large each block is. As of this writing the Bitcoin blockchain is roughly 199 gigabytes in size while Ethereum is 629 gigabytes according to [4]. It is worth noticing that Bitcoins chain started in 2009, while Ethereums started in 2015, which means that Ethereums chain grows at a higher rate. The reason for this, is that in Bitcoin it is expected that a block occurs every 10 minutes, while Ethereum it is every 14 seconds [10]. The benefits of generating blocks faster is the response time of the system, however the trade-off will be the chain growth speed.

2.1.2 Inconsistency

Consistency is required as it ensures actions performed by one node over a distributed storage system does not violate the actions performed by others. If there is no consistency control in a distributed storage system several issues can occur:

- **The lost update problem:[34]** Node 1 finds a new block (see section 2.3.2 on block mining). However before this block is distributed, node 2 finds a new block as well. Since node 2 has not seen the block made by node 1 it does not take the changes into account. One of the new blocks will therefore be lost.

- **Dirty read problem:[34]** This problem occurs when a task is allowed to read a value while another task is performing write on the same value. If this write operation is reverted/aborted the value read by the first task is wrong. In the usecase of Blockchain this problem does not directly exist since transactions in a block cannot directly be reverted or aborted. However, a new block can occur during the reading task that changes the value, but since the node will read from an old block, its value will be obsolete.
- **The incorrect summary problem:[34]** This problem relates to performing summation tasks where one or more task are impacted by the dirty read problem. In blockchain an example of the could be a contract (see section 2.3.1 on smart contracts) that uses other contracts to perform a task. If any of these contracts are modified during the execution of the master contract the outcome can be invalid. This is also known as The Inconsistent Analysis Problem.

In blockchain inconsistency can be experienced when two or more nodes have different copies of the chain. This is called an accidental fork and happens when multiple synchronized nodes discover independent blocks for the next block slot almost simultaneously. In this report any reference to forks in blockchains refer to accidental forks, unless otherwise specified. Forking is explained in detail in Chapter 13.

2.2 Blockchain Elements

The basic elements of the blockchain is seen in figure 2.1. The elements in the figure is described in the

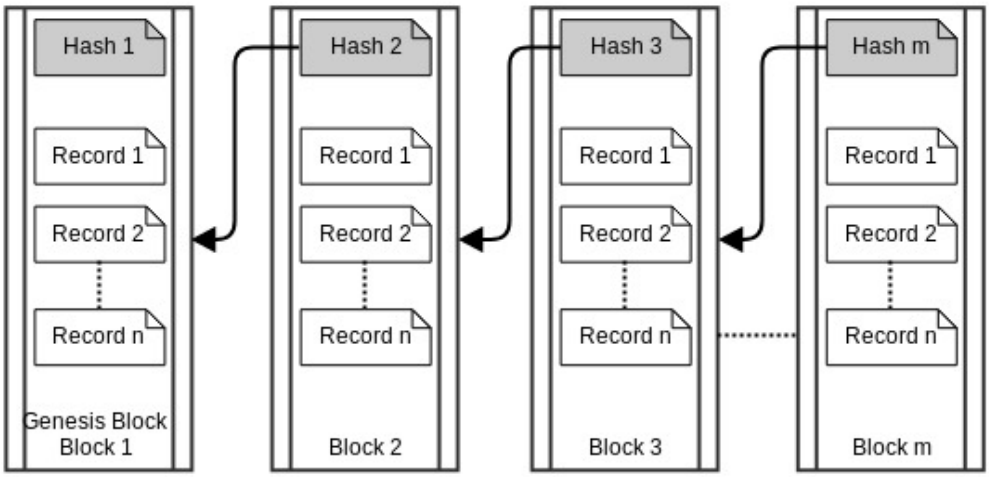


Figure 2.1: The basic elements of a blockchain.

following list:

- **Genesis Block:** The first block in the chain and is defined by the chain creators. Here the parameters of the chain is set and the creators can allocate currency to specific accounts. Things such as the initial gas limit and mining difficulty can be set (see section 2.3.2).
- **Blocks:** Blocks are sealed records and also contain the hash of the previous block in order to prevent blocks from being modified after being sealed. Every time a block is mined it can increase or decrease the mining difficulty and gas limit of the chain.
- **Records:** Records can be transactions between two or more users or smart contracts. Transactions are transferring of data and does not necessarily mean blockchain currency.
- **Hash:** Hash functions can be seen as a fingerprint of a file. No matter the size of the data, a hash function produces a fixed sized output. The hash function used in blockchain are one-way functions with strong collision resistance, which mean it is computational infeasible to find any pair that generate the same hash output.

Blockchains contain distributed trust and is obtained by creating a P2P network for every chain. When a new block occurs in the chain, the majority of members in the P2P network will have to agree if the block is valid before it can be sealed. Blockchain uses public key encryption in order to sign a transaction. If Alice want to send a transaction to Bob, Alice will use her own private key to sign the transaction and broadcast it to all her peers, see Figure 2.2a. In the transaction Alice will place her public key and in that way all other nodes in the system can validate if it was indeed Alice that made the transaction, this is seen on Figure 2.2b. Today, several projects exits based on the blockchain technology, where one example is the Ethereum

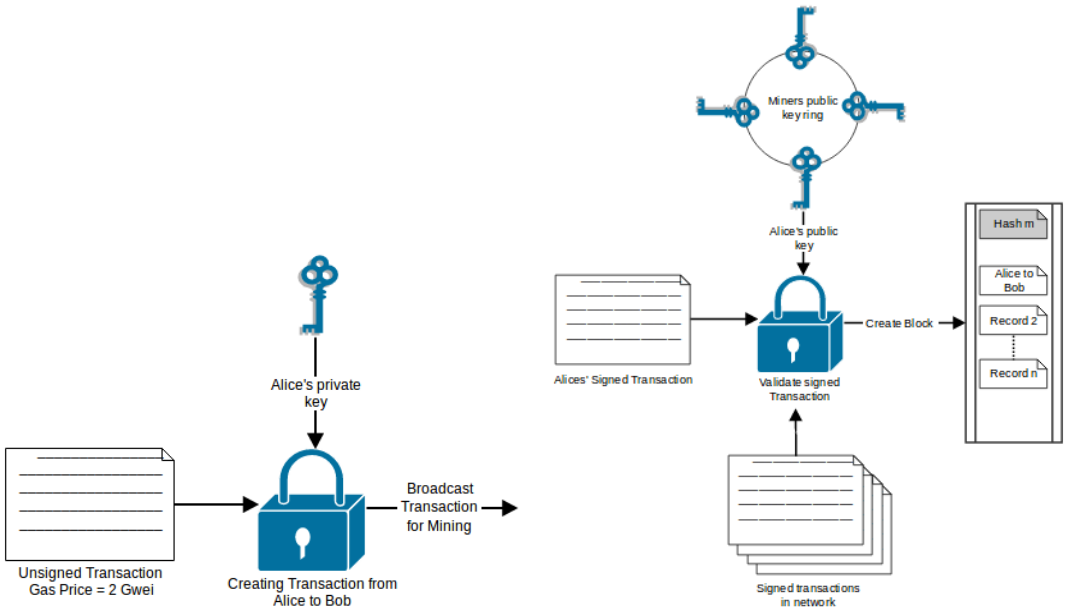


Figure 2.2: (a) Alice creates a transaction (b) A miner creates a block with Alices' transaction

blockchain, which has been of interest for the BIG IoT research project. This is due to Ethereums goal of providing a blockchain platform that is not only directed towards currencies but also towards utilizing the blockchain technology to develop entire systems.

2.3 The Ethereum Blockchain

While Bitcoins main usecase is to be a crypto-currency, Ethereums goal is to be a multi functional blockchain. Ethereum is designed for creating applications utilizing the blockchain technology and this is made possible by the Ethereum Virtual Machine (EVM). The EVM contains the functionality to create a private P2P network and a private chain on that network. The developers can create their own decentralized applications (DAPPS) upon that chain. To make it easier for the developer, there exist multiple Ethereum clients that the developer can use in order to build their application around. The Ethereum protocol exist in different implementations and includes clients written in different programming-language, amongst those are:

- **Geth:** Geth stands for Go for Ethereum, and is a client based on the go-programming language
- **Pyethapp:** Stands for Python Ethereum Application and is a Python based client
- **cpp-ethereum:** Is a Ethereum client written in c++

The clients make it possible for the developers to choose a language to their own liking. This is purposely done as it should help make it easier to get started to develop an application on the Ethereum blockchain. Every client implements the EVM and all the necessary blockchain actions, such as:

- Create new account

- Create new P2P network (private or public) and a chain
- Start mining
- Create transaction
- Create smart contract

The Ethereum clients are used to create, run and interface to the EVM. An overview of the DAPP infrastructure can be seen on Figure 2.3. In order to become a node in the network, the user needs, first of all, to

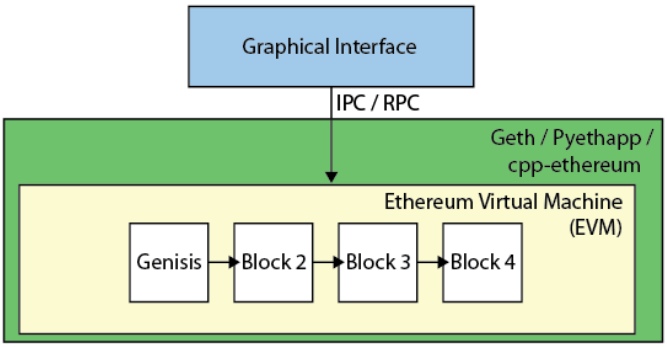


Figure 2.3: Architecture of a decentralized application using Ethereum. Illustration originated from [37].

run one of the Ethereum clients, but also the JSON file that contain the Genesis block description. When connected to the network, every node can make transactions, smart contracts and view the content of the chain. An illustration of a small P2P network can be seen on Figure 2.4. One of the useful tools to create

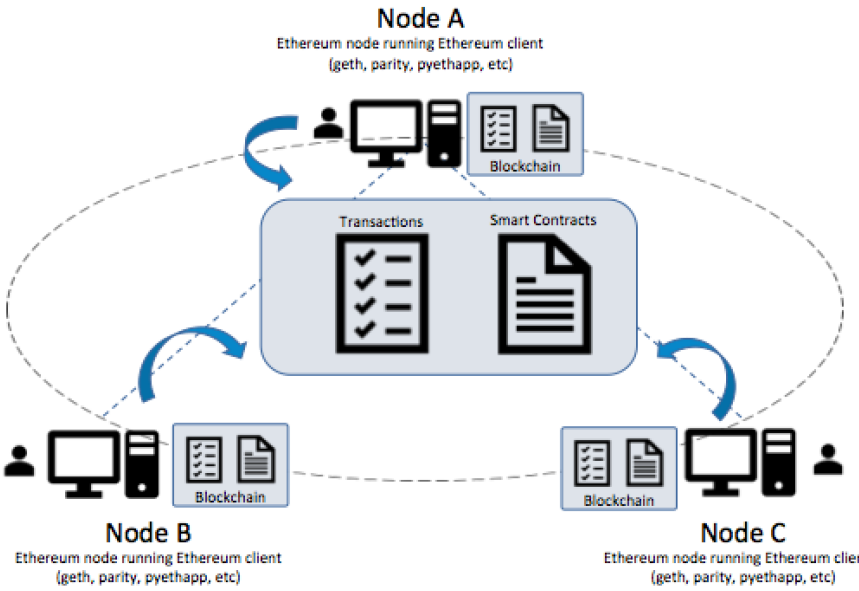


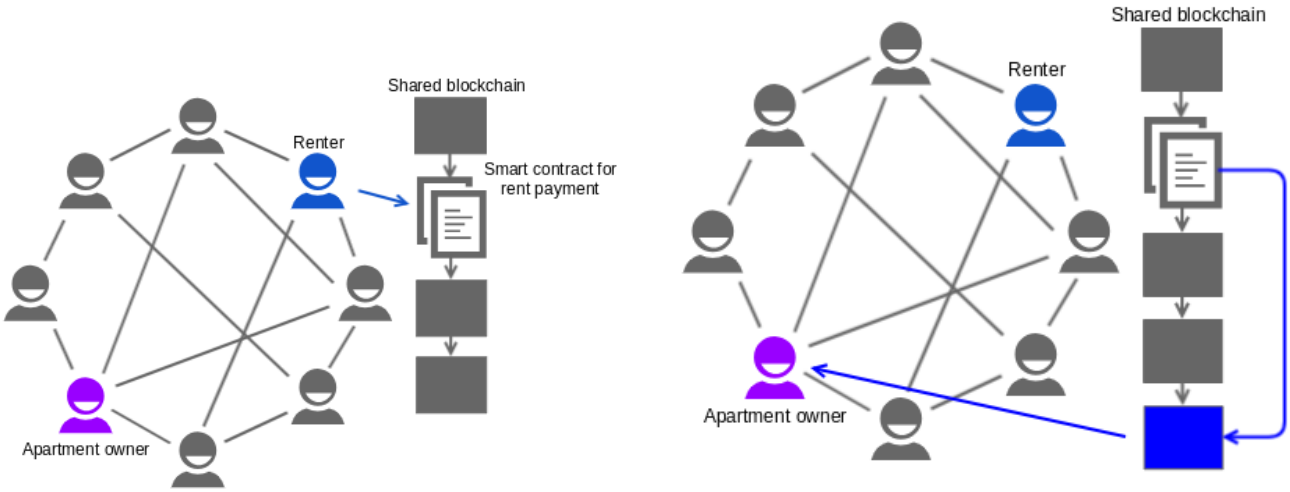
Figure 2.4: Architecture of a Ethereum blockchain P2P network. Illustration is from [5]

DAPPS is the creation of smart contracts. These allow users to create code, which can then be executed on the chain. With smart contracts it is possible to deploy other functionality and thereby create decentralized applications.

2.3.1 Smart contracts

The general idea of blockchains is to have a decentralized system based on distributed trust and thereby remove the necessity of third party involvement. However, third parties, like banks, often offers several services for its customers, everything from lending money to making periodic payments for rent, utilities or loans. To

achieve such a functionality in blockchain, smart contracts can be used. In a given scenario of a payment, the rules of a transaction is stated in the contract and distributed across the blockchain. A simple example of a usage of smart contracts could be monthly payment where the contract is created to make transactions on future dates. This can be seen on Figure 2.5. In Figure 2.5a the renter transfer money to the contract and in Figure 2.5b the contract pays the apartment owner when the rent is due. This is possible because smart



(a) (b)

Figure 2.5: (a) The user calls the smart contract inside the chain and the contract will hold his money until it is time to pay rent. (b) The smart contract creates a transaction towards the owner of the apartment when the criteria of the contract is met.

contracts are created as users on the blockchain and just like a user would have an address and a balance, so will a contract. Whereas an account can best be described as a human-based user, a smart contract is a code-based user. The programming language used to create smart contracts is Solidity, and is invented by the Ethereum group directly for Etherem. A more detailed description of Solidity is seen in section 5.3. Any action that can be performed with human interaction can be performed by a contract. Furthermore smart contracts can be used to store information which is then available to all or a subset of users. Since smart contracts are located on the chain, they undergo the same transparency and security that any other block on the chain would have. This means a contract will forever exist once it has been deployed. As contract are contained in blocks they cannot be deleted from a chain once they are committed. So the only thing a user can do to remove the contracts functionality from the chain is to include a method to disable them.

As the smart contracts are visible for all users on the blockchain, any potential vulnerability or security hole can be exploited by malicious attackers. As the blocks cannot be changed once committed, the potential vulnerabilities within the contract cannot be patched and once a vulnerability exists it will remain. This kind of attack is something that had happened on the main chain on June 2016 [16]. Here a vulnerability was found in a smart contract allowing hackers to steal 50 million dollars worth of Ether, which is the currency in Ethereum. If a vulnerability in a contract is to be solved the following steps are required:

1. Disable current contract (call self destruct function in contract)
2. Patch vulnerability in contract source-code
3. Commit new release of contract to the chain
4. Inform users of the contract of the new contract

Fixing a vulnerability in an existing contract is not an easy task. It is therefore highly desirable to ensure that a contract is properly designed, implemented and tested on a private test chain before committing it.

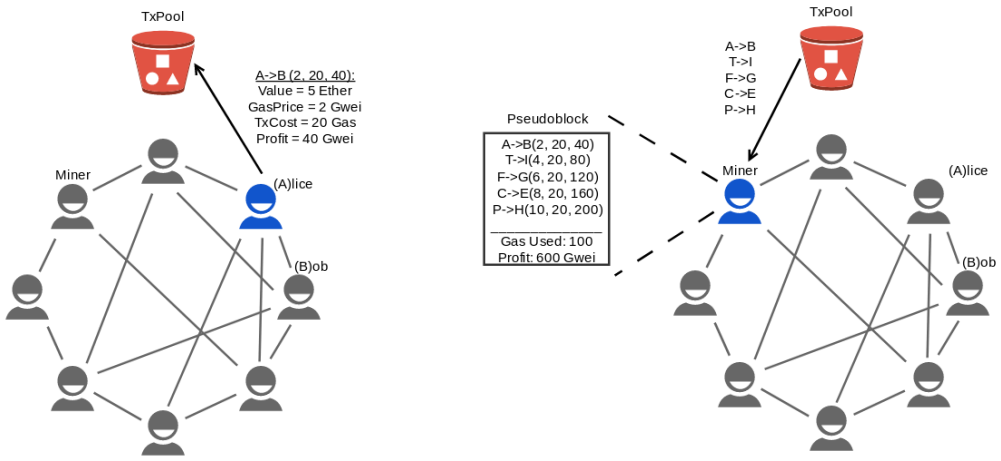
2.3.2 Mining and block creation

Mining is the operation in order to create new blocks. Whenever blocks are created, the majority of nodes in the network must validate them. When Alice makes a transaction, it will be broadcasted to all nodes in the network and placed in their transaction pool (TxPool). Simply put, the TxPools are transactions queues and can be different from node to node, as it is possible for a node to have heard a transaction before its neighbors. TxPools are not necessarily FIFO queues, but can be prioritized based on the value of a transaction. In order to understand how miners choose transactions for a new block, there are three variables that should be known:

- Block gas limit (the maximal gas that a block can contain)
- Transaction gas cost/limit (must be lower or equal to the block gas limit)
- Gas price

The gas cost of a transaction or deploying a smart contract can be calculated by looking in the fee schedule, see appendix A, which contains 31 different operation costs. The more complex a transaction is the more gas it cost to validate and the harder it can be to predict the gas cost. Therefore Alice can set a transaction gas limit which she can choose to be a number large enough to be sure that it will validated and the gas that is not spent will be returned. The block gas limit is not constant and when miners find a new block, the miner will (based on an algorithm) change the gas limit. All algorithms and equations in Ethereum is stated in the Yellow Paper [12]. The TxPool will contain transactions and many of these transaction are most likely to have the same transaction gas cost, so how does the miner choose transactions? This is where gas price comes into the picture and is also called the mining speed and refer to how attractive a transaction is.

Lets assume that Alice creates a transaction of five Ethers to Bob that has a transaction gas cost of 20 and Alice is willing to pay a gas price of 2 Gwei (Gwei $= 10^{-9}$ Ethers), the total transaction cost for Alice is 40 Gwei ($TransactionGasCost \cdot GasPrice$), this is seen on Figure 2.6a. Lets also assume that the block gas limit is 100 gas, which means that a block can contain 5 transactions like Alices, it is now up to the miners in the network to collect transactions which will add-up to the block gas limit. The transaction gas limit should match the minimum calculation required to validate the transaction, otherwise it cannot be validated and Alice will still pay for the calculations used. When the miners creates a block, they will try to create them with a total amount of gas closest to the Block gas limit and with transactions with the highest gas price, this is seen on figure 2.6b. So mining is the term for creating new blocks and is done by nodes in the



(a) (b)

Figure 2.6: (a) Alice creates a transaction to Bob that is sent to the TxPool (b) A miner collects the transactions that gives him the most profit

network. The miners in the network compete against each other and only the miner that successfully mines

a whole block will get paid the gas fee. In order for transaction not to starve, there is a requirement that miners must pick a percentage of legacy transaction. A transaction becomes a legacy transaction once it has spend a certain amount of time waiting to be mined regardless of gas fee. Likewise, to mine new currencies, miners who successfully mine the block are paid an additional amount of currency.

The additional amount gained from mining blocks slowly decreases according to the scheme in the chain until it it reaches zero at which point the currency gained from mining is only the fee paid by the transferring accounts. The discovery time between each block varies from the blockchain implementation and in Ethereum the expected block discovery time is set to 14 seconds[10]. This is based on the difficulty and the higher difficulty the longer time it will take for a miner to mine a block. The difficulty determines the amount of calculations which are, on average, required to find a block. A miner will, when trying to create a block, attempt to find a value of a hash string which falls into a region of acceptable values. As the difficulty increases the size of this acceptable region decreases. This method of mining blocks is called proof-of-work. This concept builds upon the fact that it is computationally infeasible to calculate new hash values for blocks and thereby protection against modifications of blocks. However the validation of blocks which other nodes must perform when receiving the blocks only requires to verify the block given the solution the miner found. A task which can be performed with ease. Whenever a miner has successfully mined a block, the miner can decide to increase or decrease the difficulty of the next block, if the mining of the block was too fast or too slow compared to the expected rate. The rate of new blocks will not be dependent on the numbers of miners in the network as the difficulty constantly adjusts depending on the time it takes to mine a block.

2.3.3 Benefits and trade-offs

Ethereum builds upon the core principal of blockchain. The creators of blockchain, Satoshi Nakamoto, claims that blockchain provides secure transactions that are computationally impractical to reverse and also protects buyers [22]. Translated into the BIG IoT terminology, this means the blockchain could provide a secure solution for the offering and subscription technologies. The Ethereum Foundation also states that applications using their solution can run without any possibility of downtime and third-party interference [11]. Right now the marketplace acts as a centralized server and is a single point of failure for the whole BIG IoT setup, therefore moving the marketplace functionalities to the blockchain could remove this single point of failure. Ethereum has stated two possible issues with their current implementation: The ever increasing blocksize [25] and what would happen with high amounts of transaction per second [36]. Both questions revolve around the problem of scalability and are both only at the discussion phase, which means that no solution has been found yet. It must therefore be assumed that the security and availability gains achieved with blockchain comes at the cost of performance. As blockchains are built around the concept of transactions the limiting factor is the mining operations. Currently mining is a slow and computationally difficult task, which is also why the security regarding blocks is good. However as it takes times to mine blocks it also impacts the amount of transactions the system can handle.

2.4 Initial Problem Formulation

The BIG IoT system allows for users to adapt to a single type of API, which can then be used to interact with the system. This increases the opportunities for companies to share, sell or buy IoT data. When a company wants IoT data, they can search through the offering descriptions to find the specific type of data they are looking for. Currently this system contains several different tasks necessary to perform the desired actions. This includes access management and exchange of data between Providers and Consumers. Since the BIG IoT system is a part of research project, it is always in constant development.

As marketplace plays a central role in the system, its reliability, security and scalability is therefore worth trying to improve. If the marketplace is not available it is not possible to buy or sell the data and the whole system falls apart. Since blockchains are decentralized, they are supposedly superior to centralized solutions in terms of availability and it was shown that it might also provide extra security to the system.
It is therefore relevant to investigate if blockchains could solve some of the same use-cases the marketplace does, and how that would impact the system. As presented in Chapter 2 a promising blockchain technology for creating distributed application is the Ethereum blockchain. The initial focus of this pre-analysis is therefore:

Can the Ethereum blockchain be used to handle functionalities offered by BIG IoT and the BIG IoT marketplace?

Chapter 3

Ethereum in BIG IoT

While there are many ideas and concepts being thrown around as examples of what blockchain can be used for, the reality is that very few implementations exists. Some examples of using blockchains (not necessarily Ethereum) is seen from large companies, where one example is that Maersk and IBM in cooperation has founded a new company with focus on blockchain. The idea is to use blockchain to monitor cargo across international borders and to improve global trade [19]. Another example is the The Roads and Transport Authority in Dubai that look into the use of blockchain to monitor the delivery status of every single part of a vehicle [1]. Every car will have its own chain, containing a digital ledger of its part and if a part has been repaired or replaced. For the most part DAPPS are research or play implementations, for example small games and gadgets. A collection of different Ethereum DAPPS can be seen in [8], which is a site for sharing DAPPS projects and as of this writing it has 1267 entries. However, the truth is that while blockchain has received a lot of attention there is little concrete use of the concept. Especially outside of the monetary aspects of crypto-currencies it is difficult to locate any actual implementations worth mentioning. Even though the state of the art is more or less absent, the inventors of Ethereum claims that it is suitable for a lot more than crypto-currencies. Previously in this report the idea of using blockchains inside the BIG IoT marketplace was discussed. However, since no knowledge of best practices of how blockchain networks look and how the system architecture should be, it is firstly required to determine how the architecture of the system should look. The first usecase is therefore to investigate how the actual system would have to be connected for blockchaining to work in the BIG IoT setup.

The rest of this chapter will focus on what is required in order for Ethereum to be used in BIG IoT, how it can solve different use-cases and what can be expected when doing so. The focus will be on the access management and the exchange part of the BIG IoT as this is concerned with the core functionality of the system.

3.1 Combined Architecture

The end users of the BIG IoT, described in Section 1.1, are users that access the IoT data with e.g. mobile applications. As written in Section 2.3 a blockchain, when containing many transactions, take up a lot of space. As of this writing the main chain in Ethereum takes up 629 gigabytes of storage and this is far more than can be expected for the end user to have on their mobile phones. While off-chain transactions are possible it is not possible to read and validate the state of the chain without having a full local copy of the chain. Therefore the architecture of the BIG IoT marketplace build upon an Ethereum chain must exclude the end users in the chain. This can be done by creating Consumers and Providers as nodes in the P2P network, sharing the chain and letting the end user connect to these nodes. The end users and nodes can then share the necessary information in order for the end user to interact with the chain. The proposed architecture for such a solution, and what will be used in this project, can be seen on Figure 3.1, where elements inside the red square have a full copy of the chain and helps distribute changes, while nodes outside does not have access to the chain (refereed to as foreign nodes). The architecture introduces the following elements:

- **Consumer Nodes:** Consumers are Ethereum nodes that can interact with and modify the chain. They perform Search queries and subscription actions.
- **Consumer Client:** Consumer Clients are the end users of Consumer nodes and each Consumer

should make sure to have resources to support their clients. This replaces the BIG IoT Consumers in the current BIG IoT system.

- **Provider Nodes:** Providers are Ethereum nodes that can interact with and modify the chain. Their primary task is to create offerings and manage their endpoints. These nodes replace Providers in the current BIG IoT system.
- **Provider Client:** Provider Clients are Endpoints (EP) of the Provider. This is where a Consumer client will retrieve data from.

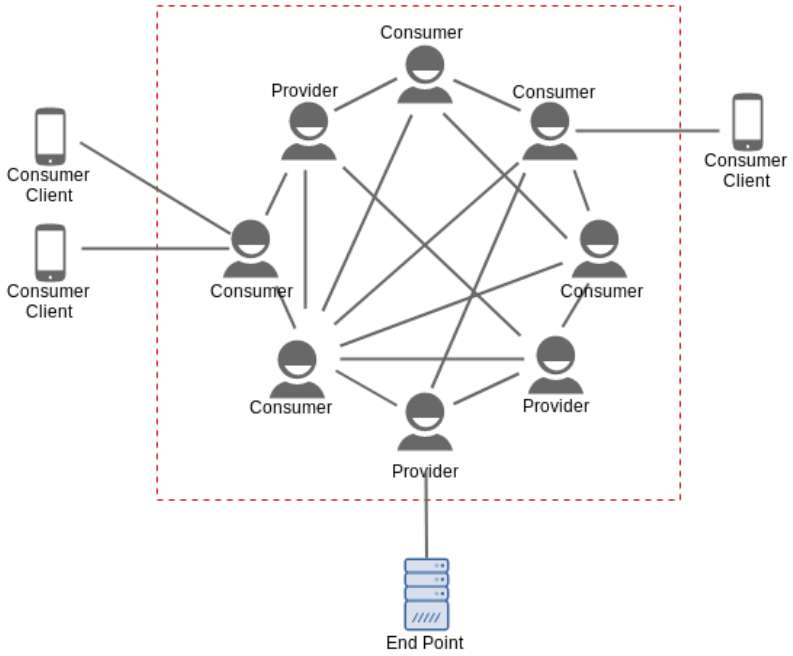


Figure 3.1: Architecture of the BIG IoT marketplace build upon Ethereum blockchain.

Right now, when the Consumer clients want to search and subscribe to an offering the end user contacts the marketplace. This means that if the marketplace server goes offline, all the BIG IoT functionality will break. Therefore this proposed architecture increases the availability on the BIG IoT marketplace since only a subspace of end users will experience an unavailable service if a Consumer goes offline. Since each Consumer must have enough resources to support their own users the total resources in the marketplace will also increase. One thing that could be an issue with this new architecture is the access to the Endpoints. Right now the end user will acquire an access token through an authentication server when subscribing to an offering. In this initial architecture an authentication server is not present and this means that the end users cannot access the Endpoint. This problem is addressed in Section 3.2.

In Section 1.1 the different areas and functionality of the BIG IoT Marketplace was presented. While it is an interesting usecase to move the entire functionality of the BIG IoT Marketplace onto a blockchain, this can quickly become a large and complicated task. Therefore this project will split the task into multiple usecases and this requires investigating which functionality is able to be moved and potentially what this would require. The BIG IoT marketplace was presented in Chapter 1.1 where the different functionality of the system was presented and as written the BIG IoT can be split into three existing areas of functionality and two more planed for the future: Access Management, Exchange, Accounting, Charging and Billing.

The functionality of the marketplace in the BIG IoT marketplace documentation [3] is defined based on the following seven usecases:

1. Create an Organization
2. Create a Provider

3. Create an Offering - by providing an Offering description
4. Activate an Offering
5. Create a Consumer
6. Create a Query - by providing a Query description
7. Subscribe - one or more Offerings that match a Query
8. Get access token - for each Subscription to access Offerings

These usecases are the basis of all of the marketplace functionality and based on the architecture in Chapter 3.1 it is possible to examine how each of the specific functionalities of the marketplace could be altered to utilize blockchains.

3.2 Access Management Usecase

The access management is the login service of the BIG IoT Marketplace. It is used to validate any access attempts to the marketplace as well as linking users to organizations, Providers and Consumers. It uses the OAuth2 protocol for identification, this process can be seen on Figure 3.2. With the new concept of

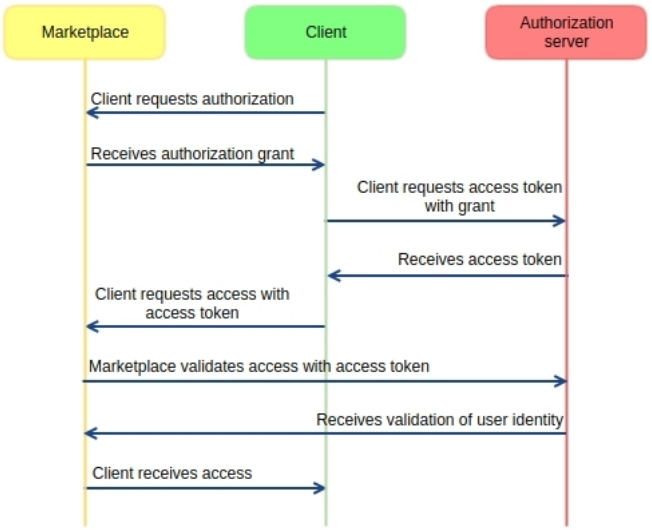


Figure 3.2: An illustration of how OAuth2 works as an authentication method in the BIG IoT marketplace.

a distributed marketplace as described in Chapter 2, the trust that the OAuth2 protocol provides will be provided by Ethereums login system, i.e. nodes that are in the blockchain are trusted. However, since Endpoints and Consumer Clients are both foreign nodes there is no existing trust between them and it is therefore required to have an authentication entity for foreign nodes. One possible solution, which is proposed by Auth0[2], introduces a centralized authentication server and a smart contract that holds user credentials. When a user is created, the user will enter a username, e.g. an email, this email will in the smart contract be associated with the users Ethereum address. Whenever the user wants access to data on a Endpoint it will contact the authentication server, and the user will need to type in his or hers password in order to get a access token. There are two main problems with this solution:

- Every Consumer Client needs its own Ethereum account.
- Since the solution introduces a centralized entity, the overall marketplace availability will be reduced to the availability to the authentication server, i.e. it will be a single point of failue

Therefore the new access management proposed in this project will be decentralized in order to keep the availability. In order to have a decentralized authentication entity all nodes in the network must possess the

authentication code, and this can be done by creating the authentication entity as a smart contract. In order for the authentication smart contract to validate if the user indeed has subscribed to an offering, it will make use of the subscription contract, see section 3.3.3. In Figure 3.3 a sequence diagram showing the steps of the proposed distributed authentication system is seen. The sequence diagram assumes that the Consumer has subscribed to the offering. When the Consumer Client (CC) request a token for a specific offering ID (OID), the Consumer checks if it has already obtained a token for that OID. If not, the Consumer will request a token from the authentication (Auth) contract, which will lookup if the Consumer has subscribed to the OID. If the status message is positive, the Auth Contract will store the token and the associated Consumer address and issue the token to the Consumer, which will then send it to the Consumer Client. The Consumer

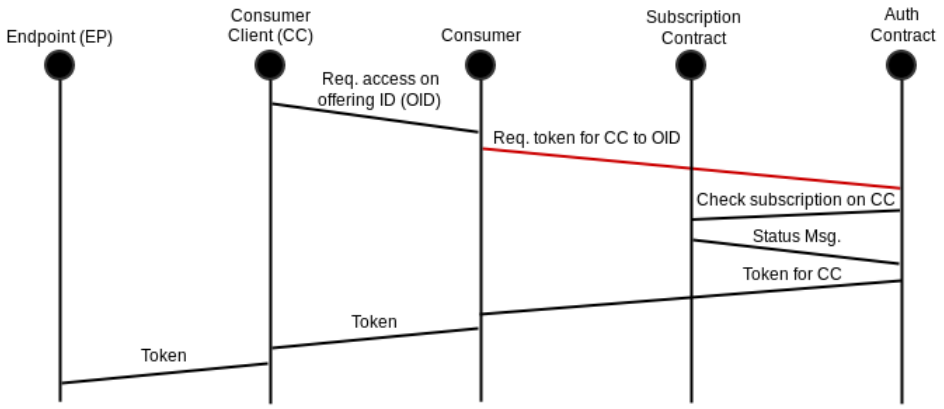


Figure 3.3: Consumer Client request a token on an offering that the Consumer has not requested before. The red line indicates that a transaction is required

Client will then request the desired data on the Endpoint with the Token and the Consumer Address and the Endpoint will ask its Provider to validate the Token, see 3.4.

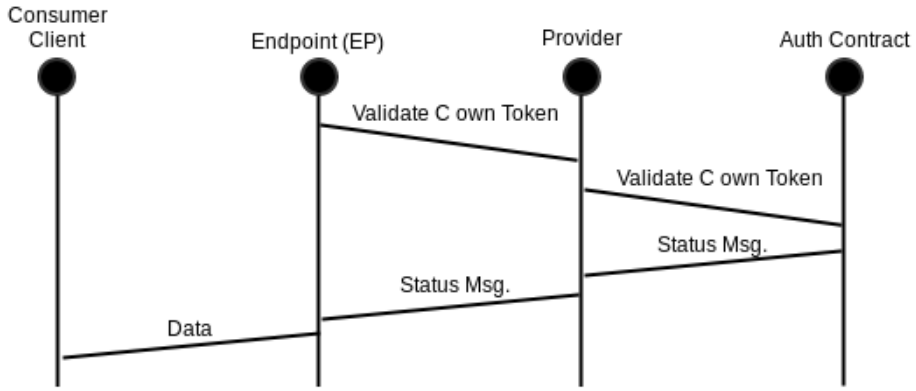


Figure 3.4: The Endpoint will ask the Provider to validate the received token based on the Consumer Address

3.3 Exchange Usecases

Exchange is the relation between the Providers, Consumers and the corresponding subscription. Currently the Exchange part of the marketplace is described by a domain model as seen on Figure 1.3. This is all managed at the marketplace, where a company / user can create an organization. An organization can then have multiple Consumers and Providers and they can then manage their content on the marketplace. The exchange functionality can be split into three additional usecases:

- Offering: A Provider can offer data, that they have collected with their IoT devices

- Offering Query: A Consumer needs to find an offering, this is done with a offering query which specifies a number of searching criteria
- Subscription: A Consumer picks one of the offerings returned from the query and subscribes to it

These tasks are dependent to each other, and the order of which is converted to blockchain is therefore important. It is for instance not possible to search through offerings if none exists. The steps for converting the exchange section must therefore be in the same chronological order as presented in the list above. In order to translate the exchange usecases, the roles in the BIG IoT exchange has to be translated into Ethereum terminology.

Organization: An organization should be able to contain multiple Providers and Consumers. Each Consumer or Provider should be able to have their own password and economy. Therefore the organization translates well to the Ethereum wallet, which can contain multiple accounts with unique passwords and unique Ether balance.

Consumer and Provider: As just mentioned, the Consumer and Provider can be translated into Wallet accounts, since they belong to a unique Ethereum wallet and have unique passwords and balance.

As mentioned there is a specific order of which different functionality should be converted to blockchains, and since it is not possible to create an offering query or subscription without any offerings, this is the logical starting point.

3.3.1 Offering Usecase

Currently the offerings are located at the marketplace, which is a centralized server where a user has to connect in order to create or search through offerings. Moving the offerings to the chain, mean that the marketplace will become decentralized and that should mean an increase in the availability. To move the offering method, the following is needed:

- The system needs a marketplace account or smart contract address to receive the offerings
- A Provider creates transactions containing all relevant offering information:
  - Price
  - Endpoint
  - License
  - Smart contract address
  - Input parameters, e.g. Consumer Client location and radius of search
  - Output parameters, e.g. lists of all parking spot in a radius and availability of the parking spots
- A smart contract could be used to verify the structure of the offering

An illustration of how offerings can be created is seen on Figure 3.5a, where a Provider sends an offering to a smart contract on the chain (1). The smart contract will then make a transaction with the offering and miners can get it from the TxPool (2). This solutions raises some issues:

- It will cost Ethers to make an offering. This means that Providers in theory can run out of Ethers and thereby be unable to make an offering
- Right now, the Provider creates offering descriptions via the BIG IoT marketplace, which means that this proposal will break the functionality of already implemented offering applications.

The problem regarding Ethers will be discussed later in this report, however it is not a usecase that will be implemented in this report. In order not to break the functionality of already implemented offering application, the marketplace could still contain the current API for creating an offering. Instead of putting the offering on a database, the API on the marketplace can be rewritten to use the smart contract instead of a database. An illustration of this is seen on Figure 3.5b, where the Provider contacts the marketplace as it would be done at the current moment (1). The marketplace then contacts the smart contract (2) and the smart contract will then make a transaction (3). This will introduce a centralized system, but on the other hand gives the Provider companies time to change their applications.

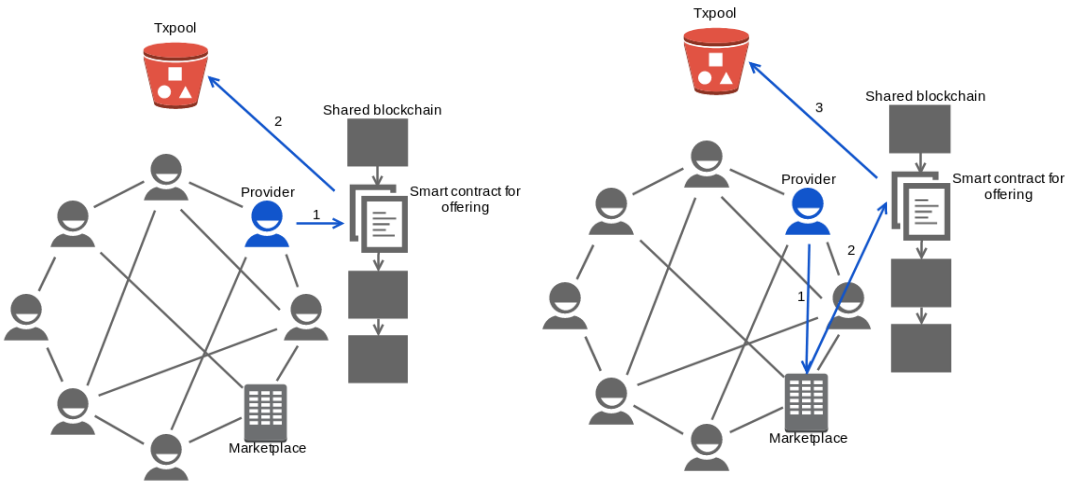


Figure 3.5: (a) The Provider creates an offering directly on the chain (b) The Provider creates an offering through the marketplace

3.3.2 Offering Query Usecase

When the offerings are placed on the chain it should be possible to search through them, as it is in the current marketplace. This requires being able to perform search queries on blocks inside the chain. To convert the query the following is needed:

- Search method for the offering chain to find valid offerings based on the query
- Return method for a list of offerings that meet the query specifications with the relevant information

The offering query will work by contacting the offering contract, which holds a method to return a given offering. The memory of where the offerings are located is not within the contract itself, instead the contract holds a reference to where in the chain the data is placed. An illustration of the offering query can be seen on Figure 3.6a, and when the Consumer request offerings from the offering contract (1), it will collect the data from where it is placed in the chain (2), and then return the actual data (3). Once again, this solution will break the current implementation, but again it is possible to make it in a way, such that the BIG IoT marketplace can be still be used, this is seen on Figure 3.6b. One thing that is crucial for a search algorithm

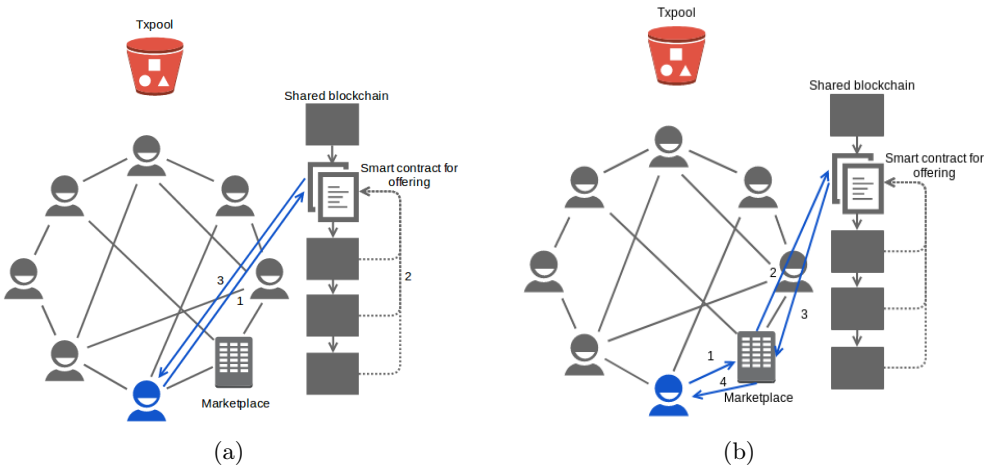


Figure 3.6: (a) The Consumer receives offerings directly from the chain (b) The Consumer receives offerings through the marketplace

is the ability to return the desired information and the time it takes to perform the search. If starting with

the time complexity, it will depend on the structure of the data. The time complexity of a search will depend on the number of offerings on the chain. This gives a search time complexity on O(n) to loop through all offerings. However, an offering is created with 5 parameters that the user can search for: License, Price, End Point, Spatial Extent and Data Field. The Spatial Extent is created with two variables: Latitude and Longitude, and the Data Field consist of both input and output data, which can be of variating size. To lower the complexity, the designer of an offering could declare a max value for input and outputs allowed in an offering, we can call the max allowed input and output for L. The total time complexity of an offering search will then be described by Equation 3.2. The $5 + L$ in equation Equation 3.2 is defined by the 5 parameters: License, Price, End Point, Latitude and Longitude, and L is the maximum size of the Data Field.

$$O(n) \tag{3.1}$$

$$O(n \cdot (5 + L)) \tag{3.2}$$

It is therefore important to examine the searching time of the offering query presented in the project.

3.3.3 Subscription Usecase

Subscribing to an offering is one of the essential features of the marketplace. Currently the subscription is set up in the marketplace where a contract between Provider and Consumer is created based on the terms of the offering (price and license). Once a subscription is created an access token is assigned to the Consumer. This token must be used any time the Consumer wants to access the Providers data. This means, in order to convert the subscription the following is required:

- Pick an offering from the returned list from the offering query
- Call a smart contract to create a subscription between Provider and Consumer

An example of how subscriptions could be performed with the help of blockchains can be seen on Figure 3.7. A Provider creates an offering on the blockchain(1). When the Consumer queries the offerings, it discovers

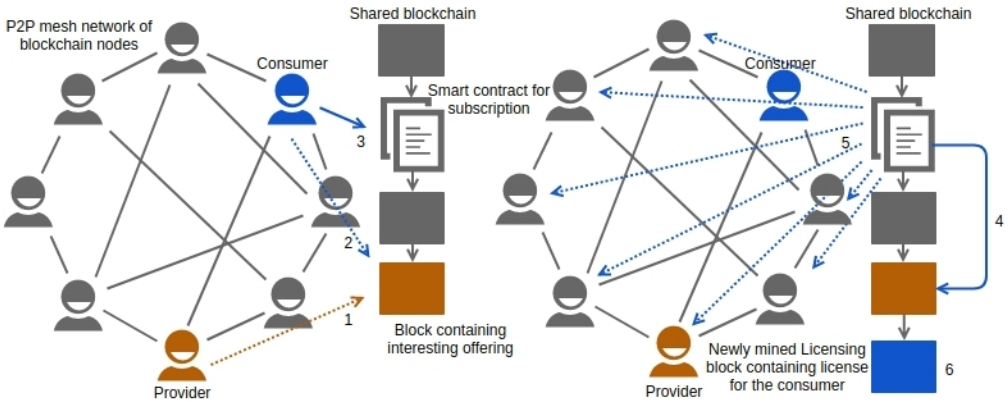


Figure 3.7: An illustration of how interactions inside a blockchain network can be used to create subscriptions between Providers and Consumers given the offering is located on the chain.

this offering(2). In order to subscribe to this offering, the Consumer calls a smart contract with the relevant information, such as what block it was found in(3). The contract can then collect subscription information from the block i.e. license type and price and creates a transaction which indicates a subscription between Provider and Consumer has been established(5). This transaction is then collected and mined into a new block, at which time the subscription is active(6). In the case the offering is not located on the chain, the Consumer and Provider might have to both sign the smart contract before a subscription is created to ensure no-one is cheating.

3.4 Summary

The initial goal of this report has been to investigate and design a new BIG IoT system based on the Ethereum Blockchain. Since Ethereum blockchain is lacking in the state of the art, it was important to clarify a solid architecture for the solution that would not break the principals of the BIG IoT system. The architecture in this project was therefore created with blockchain nodes which contained the BIG IoT operators Consumers and Providers, but furthermore introduces foreign nodes that was not a part of the blockchain. Several usecases from the BIG IoT system has been analyzed and based on this the usecases has conceptually been integrated into the new combined architecture. The functionality provided by the Ethereum smart contract was seen to be a useful tool in order to provide the same functionality in a distributed manner. It is therefore deemed possible to use Ethereum blockchain as a technology to achieve a distributed BIG IoT system.

Chapter 4

Problem formulation

As evident by Chapter 3 there are several usecases in the marketplace where blockchain can be used. Currently the marketplace contains tasks for authenticating users, issuing access tokens, creating offerings, searching the offerings, creating subscriptions and perform accounting on the use of the subscriptions. As written in chapter 3.3 there is a natural flow between offerings, offering queries and subscriptions and it therefore makes sense to work with them in the stated order. The usecase regarding access management is needed in order for subscribers to access the assets on the Endpoint. However the general idea behind subscriptions will still function without the access management and can therefore be left out. Instead of creating the access management, the functionality of foreign node access will be implemented, since it introduce an important role in the new architecture. This project will therefore focus on implementing and testing usecases in the following order:

1. Create offerings
2. Perform offering queries
3. Create subscriptions
4. Gain foreign node access

In order to determine the potential gains from using the Etheruem blockchain the usecases will firstly be implemented and thereafter evaluated. Currently there are several claims made by Ethereum in regards to their blockchain. In chapter 2.3.3 they claim that the chain increases security and availability, but they do however raise concerns with the technology's performance constraints with mining, when the numbers of transaction increases in the network. Mining will therefore impact the time it takes to create, delete and modify offerings and subscriptions. It is also relevant to investigate the searching time of the offering query as the number of offerings grow. Furthermore as blockchain is a distributed technology the problem with inconsistency is inevitable. This report will therefore focus mainly on the following four areas:

- Time aspects
- Security
- Inconsistency
- Availability

This report will therefore focus on the previous four areas and they will be further explained in Chapter 10. The rest of the report will therefore focus on the following problem statement.

4.1 Problem statement

How can the BIG IoT usecases be implemented using the Ethereum blockchain? What advantages or disadvantages will it introduce in these usecases?

Part II

Implementation

Chapter 5

Implementation Platform

To implement the discussed usecases it its necessary to have a setup which allows them to be implemented. In order to create a P2P network where all peers share the same blockchain, one of the implementations of Ethereum is needed. Currently Ethereum is implemented as Geth, Pyethapp and cpp-ethereum. It does not matter which implementation is used as they all provide the same functionality. Since Geth was the first Ethereum implementation that was successfully tested in this project, it is the implementation chosen for this project.

5.1 Geth

With Geth it is possible to create a private Ethereum chain, a private network, accounts and makes it possible to perform all the necessary Ethereum actons. To interact with Geth there is a JavaScript command-line tool, where it is possible to import custom made JavaScripts. In order to have a working private chain in Geth there are several steps to consider.

5.1.1 Chain and Node Creation

In order to create an Ethereum chain, Geth needs to know the location of where to store the private chain, how to initialize the first block in the chain and what the node should be called. This is done with the command seen in Code-snippet 5.1.

Code snippet 5.1: Command to create private chain

```
geth --identity "node_name" init genesis.json --datadir "chain_location"
```

The location of the local stored chain is specified by the datadir tag and this is where all future blocks will be downloaded to. The name of the node is not a required input, but can help to make the network more manageable. The initialization of the first block (the genesis block) is defined by the genesis file, an example can be seen in Code-snippet 5.2.

Code snippet 5.2: Example of a genesis json file

```
{
    "config": {
        "chainId": 1234,
        "homesteadBlock": 0
    },
    "difficulty": "20",
    "gasLimit": "2100000",
    "alloc": {
    "5f3f76425680ffe93a7a469a9bcf978b0cb85fba":
    { "balance": "300000" }
    }
}
}
```

As long as the genesis block is the same at all nodes, they can always re-sync their chain if the local version was deleted. The Genesis block contains the following parameters:

- **config:** The configuration identifier for the chain. What lies inside defines the chain itself.
  - **chainId:** Every chain should have its own unique identifier (id). The main chain in Ethereum has id = 1. To avoid collisions with other chains the id should be sufficiently long and could be chosen randomly.
  - **homesteadBlock:** Ethereum version identifier. As for now, Homestead is the newest version of the Ethereum platform and the first production build to be made [6]. Since its already using Homestead, the value of homesteadBlock can be set to 0.
- **difficulty:** Initial blockchain difficulty. This value will be adjusted when blocks are mined, depending on the amount of calculation power in the network
- **gasLimit:** Initial gas limit. The aggregated gas of all transactions in a block must therefore be below this number.
- **alloc:** This will allocate predefined accounts for the creator of the chain. This can be left empty, if no accounts should be created in the creation of the chain.
  - **balance:** Is used if the creator of the chain wants to fill an account with Ethers without having to mine it.

After the initialization of the chain with the genesis block, it is possible to create a private network that will share this blockchain. In order for nodes to join the chain, they too need the genesis block. This is the only time where nodes cannot share data across the blockchain but has to be shared by some other method. This means a node which wants to join a specific chain, must have access to that chains genesis block. In order to create and start the network, the command shown in Code-snippet 5.3 is used.

Code snippet 5.3: Command to create private network

```
geth --datadir "chain_location" --networkid "some_integer" --rpc
```

Upon creation, Geth creates an Inter-process Communication (IPC) location which is a pipe for the local computer to access the chain. If a user wants to access the chain remotely, a Remote Procedure Call (RPC) can be created, which works across the network. In order to create RPC access, the rpc flag must be set in the Geth call. If the rpc flag is set, Geth will create a local host RPC with default port (http://127.0.0.1:8545), in order to create a non-local host, it is possible to change the default port and address with the Code-snippet 5.4.

Code snippet 5.4: Command to change default port and IP of the RPC

```
  geth --rpc --rpcaddr <ip> --rpcport <portnumber> --rpcapi <list>
```

When starting Geth with the rpc flag, the creator needs to tell which rpcapis should be allowed for the remote call. This is done by setting the rpcapi flag, as seen in Code-snippet 5.4, followed by a comma separated list. The possible rcpapis are admin, db, debug, eth, miner, net, personal, ssh, txpool and web3. In order to connect to the Ethereum node, the Geth attach command can be used, see Code-snippet 5.5

Code snippet 5.5: Command to attach to the node

```
  geth attach rpc:http://<ip>
```

This command will open the Geth JavaScript Console, with access to the specified rpcapi commands. In Code-snippet 5.6 a few examples of commands can be seen:

Code snippet 5.6: Geth console command examples

```
  miner.start() // Start mining
  miner.stop() // Stop mining
  personal.newAccount() // Create new account
  personal.listAccounts() // List all accounts in the P2P network
  personal.sendTransaction({to: "account", from: "account", value: "amount"
     }, 'pass-phrase') // Send ether
  loadScript("name_of_file.js") // Execute JavaScript File
```

While allows for creation of a blockchain network, it does not provide any default solution to peer discovery, nor does it provide any Solidity compiler which is needed in order to deploy smart contracts. These two areas must therefore be investigated.

5.2 Peer Discovery

Peer discovery in Ethereum is based on the Kademila protocol defined in [20]. It utilizes four basic messages namely ping, pong, findnode and neighbors to achieve peer discovery [18]. One of the problems with Ethereum is that the default method of peer discovery does not work in private chains. In the public chain a user will, whenever they come online, use a list of so called bootnodes to find peers. Ethereum is created with a predefined list of bootnodes which a new node can connect to. Once it finds a valid node it will initiate a Ethereum handshake which, if completed, binds the two nodes together as peers. Peer discovery in the Ethereum chain can be defined as:

- Designated bootnodes are assumed to always be online
- Bootnodes maintain a list of all nodes connected to them in a given time period (example: 24 hours)
- When a node connects to the chain (launches Geth, Pyethapp, cpp-ethereum) it will connect to the bootnodes, which will return a list of connected peers and add the node to that list

Since there is no valid predefined list of bootnodes in private chains this method will not work directly. It is therefore not enough to create a chain with the same genesis and network id, as they will not be able to utilize the predefined bootnodes to find each other. Geth supports the addition of singular nodes either manually or by defining them in a static node file. However this is a slow and ineffective process as it requires knowing the exact enodeID and IP address of every single node in the network, where enodeID is the specific identifier for an Ethereum node. To solve this issue it is possible to create private bootnodes which performs the same tasks as the public nodes. A private bootnode will therefore be a separate node from the Consumers/Providers in terms of topology, but can be located on any device inside the chain. This will add an element of centralization to the network as they are required to be online at all times to ensure peer discovery. To mitigate this, the same procedure as in the public Ethereum chain can be used by creating multiple bootnodes and providing nodes in the chain with a list of their addresses. An important thing to remember for this to work is to ensure that the bootnodes are located at the same address at all times i.e. have static IP addresses and enode ID. If their addresses change the nodes will not be able to find them.

As Geth contains all the necessary tools to create and operate a bootnode it is relatively simple to set up a bootnode.

Code snippet 5.7: Commands to configure and initiate bootnodes.

```
1 bootnode --genkey=boot.key //Ensures all connecting nodes share the same
    genesis
2 bootnode --nodekey=boot.key //Hardcodes the nodeID of the bootnode so it is
     always located on the same enode address
```

By using the commands in Code-snippet 5.7 a single bootnode is created which will be able to perform initial peer discovery. To utilize the bootnode the bootnodes flag must be used when starting Geth as seen in Code-snippet 5.8. If multiple bootnodes are defined they should be appended in the command.

Code snippet 5.8: Geth initialization command with the bootnodes flag.

```
1 geth --datadir "chain_location" --networkid "some_integer" --bootnodes <
    bootnode enodeID from above>
```

Whenever a Geth instance is launched with this flag it will automatically connect to the defined bootnode to discover peers. This solves the problem with peer discovery in private chains. By ensuring peer discovery it is possible for any node with a link to a miner to have their transactions mined. This ensures that nodes can always utilize the smart contracts on the chain.

As written, the smart contracts are developed in Solidity, but Geth does not come with a Solidity compiler. Luckily this problem has already been addressed and there exists several tools to deploy smart contracts using the IPC or RPC in Geth. One of the most popular tools are Truffle and is the one used in this project.

5.3 Truffle

Truffle is a framework that can be used to create a smart contract in Solidity and then compile and deploy them. Truffle comes with a number of functions:

- **truffle init:** Initialize the Truffle project and creates the initial migration contract, initial migration script and a truffle setup file.
- **truffle test:** Tests all the contracts in the contract folder. Returns passed or failed on every contract.
- **truffle compile:** Compiles all the contracts and outputs all the compiled contracts to a newly created build folder.
- **truffle migrate:** In order to deploy a contract, migrations are used. When initializing the truffle project, it is created with a initial migration contract. As written in the truffle documentation: "Truffle requires you to have a Migrations contract in order to use the Migrations feature"[33]. When calling truffle migrate, Truffle looks inside the truffle setup file in order to see which network to use, this will be a Geth rpc url. The initial Truffle migration contract will only deploy the initial migration file once and will not require it to be updated again.

As written, the smart contracts are written in Solidity, which is a high level language directly created for EVM. Solidity is a contract-oriented language and implements the following data types:

- Datatypes
  - **Integers:** uint / int
  - **Booleans**
  - **Address:** Ethereum address (20 bytes).
  - **Address members:**
    - **Transfer:** Transfer Ethers to a given address. If the sending account does not have enough funds, the function will return false.
    - **Balance:** It is possible to check the balance of an account in a contract, and transfer Ethers if the account balance meets a certain requirement
    - **Call:** In order to interface with a deployed contract, the call function can be used.
  - **Strings**

Solidity also support return functions that can return one or multiple of the data types. A function can be marked with different visibilities:

- **External:** Can be called from other contracts and transactions. However, it cannot be called by functions within the contract itself
- **Public:** Can be called from other contracts, transactions and internally from within the contract
- **Internal:** Can only be called from within the contract itself or contracts deriving from it
- **Private:** Can only be called from within the contract itself

Now that it is possible to create a node and a chain using Geth, it is possible to create the architecture described in Chapter 3.1 and to implement the different use-cases.

5.4 Testbed

Now that all necessary tools has been developed, it is possible to create a testbed to both implement and later test on.

5.4.1 Architecture

As written in Chapter 2.3.2 the number of nodes in the network does not change the mining speed of blocks and the testbed architecture can therefore be simple. The architecture, seen on Figure 5.1, consists of three nodes that share a private chain. The nodes have the following setup:

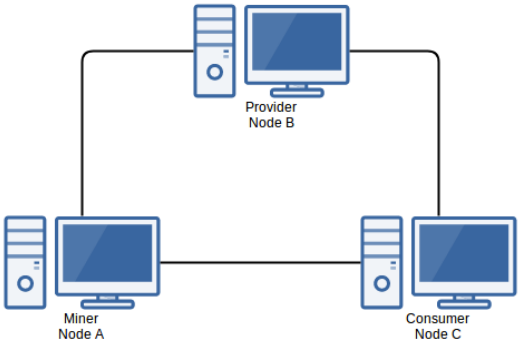


Figure 5.1: Architecture of testbed

- Ubuntu 16.04
- Geth version 1.8.10
- Node.js version 8.11.1
- Truffle version 4.1.0
- npm version 5.7.1
- Docker version 1.13.1
- Docker-compose version 1.8.0

Node C also contains the Apache and PHP configuration as seen in Section 9.1

5.4.2 Testbed parameters

When the testbed is used for testing purpose, it is necessary that all test have the same base in order for them to be compared. This is done by performing each test based on the same genesis file with a predefined difficulty and gas limit.

Difficulty

As the block mining rate represents the systems responsiveness it is important that the rate experienced in the testbed is as close to the real world scenario as possible. As mining blocks is not an instant process it is necessary to investigate if the mining of blocks is as expected in the test setup. This will both impact the end result if form of system responsiveness but also the tests following. The mining rate of blocks can be theoretically estimated, as block mining time is exponentially distributed [27]. The theoretical mining rate of blocks is described as a continuous random variable X with PDF, see Equation 5.1.

$$f_X(x) = \begin{cases} \frac{1}{14} \cdot e^{-\frac{1}{14} \cdot t} & for\ t > 0 \\ 0 & Otherwise \end{cases} \tag{5.1}$$

In order to get the expected block mining time, the difficulty must have been stabilized in the testbed. It is therefore required to determine at which difficulty the testbed will, on average, generate blocks every 14 seconds. In order to test the difficulty needed in the testbed, a large number of blocks are mined. The difficulty will be extracted from where the mining time stabilized. As written in Section 5.1, it is possible to extract data from the chain by creating JavaScripts. Since JavaScript itself cannot make Geth commands, the web3.js library is needed.

Web3

Web3 is a JavaScript library that contains all the modules to interact with the Ethereum ecosystem, and is created by the Ethereum Group. Once installed it is possible to connect to a web3 provider through JavaScript, an example of connecting to the localhost RPC is shown in Code-snippet 5.9. In this code example it is used through a browser, the code example is using the HttpProvider.

Code snippet 5.9: Connecting to local RPC with HTTP Provider

```
1  App.web3Provider = new Web3.providers.HttpProvider('http://127.0.0.1:8545
       ');
2  web3 = new Web3(App.web3Provider);
```

Now, through the newly created web3 variable, it is possible to call all methods that the RPC allows. An example could be that it is desired to extract all information from all blocks in a chain, this could be done by the example shown in Code-snippet 5.10, which logs the difficulty, miner address and timestamp of every block in the chain.

Code snippet 5.10: Block information extraction with web3

```
1 var endBlockNumber = web3.eth.blockNumber; // Get number of blocks in chain
2 for (var i = 0; i <= endBlockNumber; i++) {
3   var blockInformaitDifficultyon = web3.eth.getBlock(i);
4   console.log(blockInformaiton.difficulty);
5   console.log(blockInformaiton.miner);
6   console.log(blockInformaiton.timestamp);
7 }
```

In many cases, it is desired to visually inspect results and for this Plotly is used, which is a JavaScript plug-in that enables creating plots in JavaScript.

The difficulty used should for the test should lead to an expected block generation time of 14 seconds and it is therefore necessary to test at what difficulty this happens in the test network used in this project, see Figure 5.1. The web3 tool is therefore used to extract the block difficulty from a test chain with more than 76000 blocks created by the test network architecture. The development of the mining time is seen on Figure 5.2, where the orange line is an moving average (MA) over 50 blocks. A visual inspection of the MA

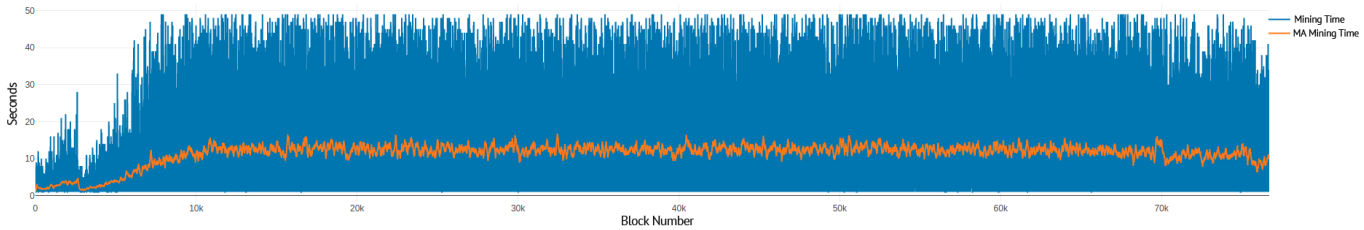


Figure 5.2: Showing the development of mining time in the testbed

revealed that it started to stabilize after block 11000, so the result, when taking an average from block 11000 to 76000 was a difficulty of $3.5 \cdot 10^7$ and gave a block generation average of 13.54 seconds.
In order to validate the newly found difficulty, a new chain that starts a this difficulty is made. A set of 1000 blocks are mined in the chain and compared to the PDF in Equation 5.1. The results can be seen on Figure 5.3. As can be seen on the figure there are deviations from the PDF, however the general trend is clear and

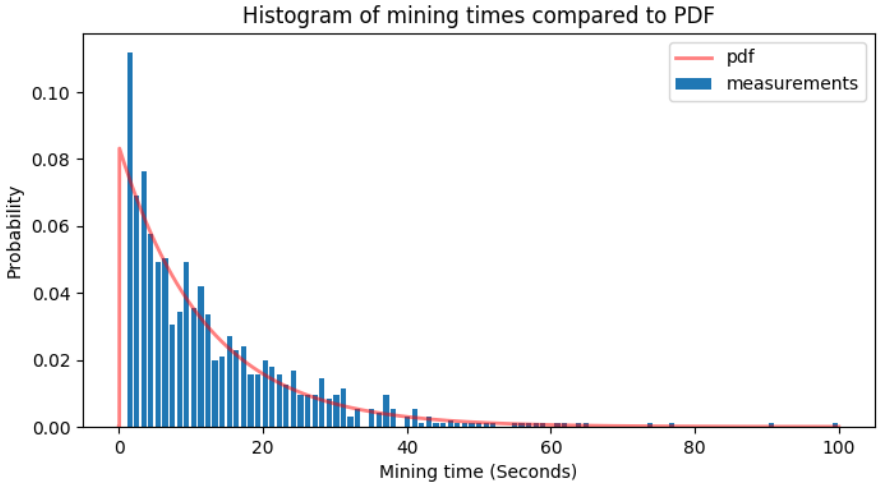


Figure 5.3: Histogram of block mining times compared to the theoretical mining distribution.

it is exponentially distributed. It is also clear that curve of the PDF follows the data nicely. This is further underlined when looking at Figure 5.4. Here it is evident that the CDF of the two is similar. It can therefore

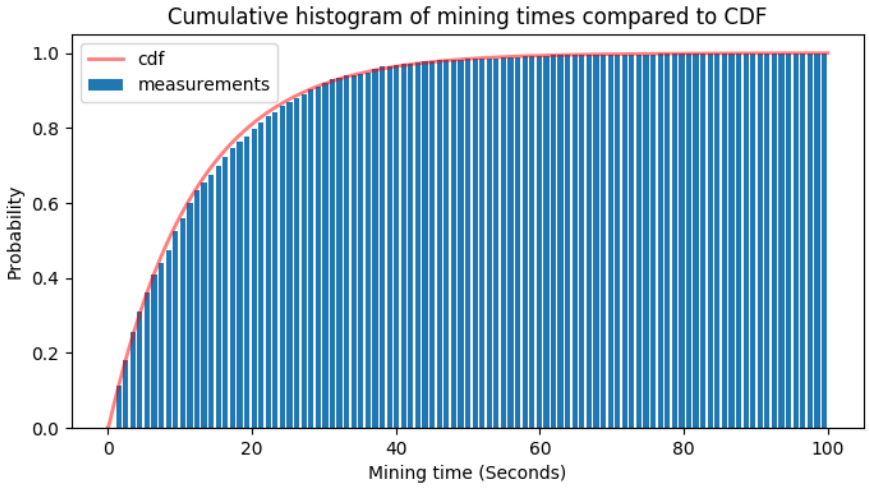


Figure 5.4: Cumulative histogram of block mining times compared to the theoretical mining CDF.

be concluded that the newly found difficulty is valid and that the mining conditions experienced in the test setup is an adequate representation of the real world.

Block Gas Limit

As written in Chapter 2.3, the block gas limit tells how many transactions that can be in an block. One of the trade-offs picking a large block gas limit is that it takes longer time for a block to get distributed. The trade-off of choosing a small block gas limit is that it will limit the number of transaction in one block. Since smart contracts needs to be deployed before each test can be made, it is necessary to have a large enough block gas limit to be able to deploy them. Currently the block gas limit on Homestead is $8 \cdot 10^6$ [9] and it is therefore the one used for the tests.

Chapter 6

Offering Usecase Implementation

To implement an Ethereum smart contract it is first necessary to identify what is relevant for the offering contract. In the current BIG IoT marketplace a Provider can create an offering by specifying parameters for the offerings such as category, price, license and what input and outputs it can be used. As the system should contain the same functionality as the original marketplace all of these areas must be specifiable in the smart contract. The general area of offerings can be split into multiple areas of functionality, each necessary in order to have a functional solution.

- Create an offering
- Delete an offering
- Look up offering
- Modify created offering

There are two types of interactions with blockchains. One can read data from a chain or one can modify the chain (add/modify/delete) chain elements. Any action which changes a chain costs Ethers as this is seen as a computational task for the mining nodes. Reading from a blockchain is a local task performed only by the users own node and therefore does not require mining. Each of these functionalities will be further described more individually in terms of how they are designed and developed but should be seen as a single offering contract entity.

6.1 Create offerings

To ensure that offerings can be created by Providers the functionality of the contract is open to all users in the chain. Any account on the chain with sufficient funds of Ethers to create a transaction can therefore create an offering. As the fields of such an offering, to a large extent, is customizable by the users, there is a need to validate the inputs given to the contract. The general functionality of creating an offering is described by the flowchart on Figure 6.1. Whenever a user calls the addOffer function in the contract it is done so with all the parameters of an offering. The license of the offering cannot currently be specified outside of the options available in the current marketplace which is: FREE, PER_ACCESS, PER_BYTE, PER_MESSAGE and PER_MONTH. If anything outside of this is specified then the offering is invalid and not accepted. If these parameters are properly set then the offering is stored as a struct inside the contract. As the offerings is a description of the data and it is vital that the format of an offering is standardized, so anyone accessing the offerings knows what to expect. To store an offering the struct functionality in Solidity is used.

```
1 struct OfferStruct {
2     // Ethereum parameters
3     uint ID;
4     address provider;
5     // BIG IoT parameters
6     string name;
7     string category;
8     uint price;
```

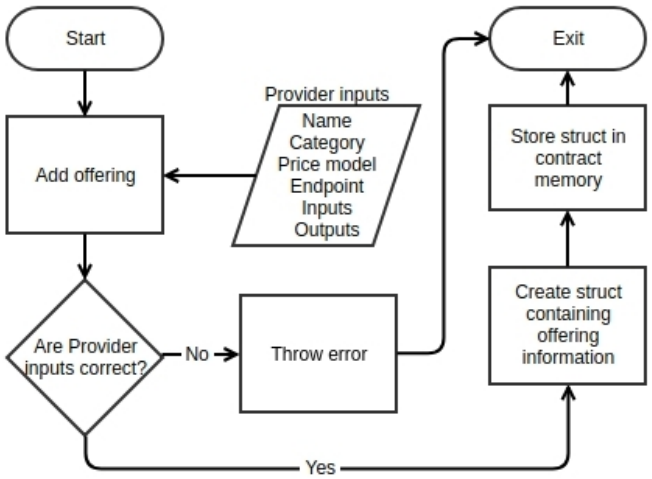


Figure 6.1: A flowchart describing the functionality of the addOffer function in the offerings smart contract.

```
9       uint endTime;
10      string endpoint;
11      bytes32[] inputs;
12      bytes32[] outputs;
13  }
```

To identify offerings a unique ID is given to each offering upon creation. This ID can later be used to request offering information or as a subscription identifier. As written in Chapter 2.3.1 Solidity is designed specifically for Ethereum and has an address variable which can only contain a valid user address. In this case the Provider will automatically be set to the account creating the offering and it is therefore not possible to set a Provider different from the account address. When a Provider wants to add inputs and outputs to the offering it is unclear as to how many they might specify and currently there is no limit to this in the BIG IoT marketplace.

Since Solidity is still in development there are some features missing in Solidity. One limitation of Solidity is that it is not possible to create 2-dimensional arrays. As a string in Solidity is seen as an array of chars, it is thereby not possible to return an array of strings. As it is possible to convert a string to the byte32 format, it then becomes possible to store them in an array. This will however limit the input and output parameters to a 32 byte size, but this is deemed sufficient. Therefore, whenever a user adds inputs or outputs they are converted from strings to bytes32 and placed in a bytearray. Reversion of this conversion is done at the offering query described in Chapter 7. One of the smart things about Solidity and the Ethereum blockchain is that libraries work by deploying them to the chain and linking them to the necessary smart contract. This mean the functionality of the library is located on another block than the smart contract. The link between them ensures that the functionality of the library can be used in the contract. In order to get an offering to the contract, a Provider calls the offering contract function addOffer.

```
1   function addOffer(string name, string category, string model, uint price,
        string endpoint, string inputs, string outputs, uint16 year, uint8 month
        , uint8 day, uint8 hour) public {
2       OfferStruct memory offering;
3       offering.ID = uniqueIdentifier;
4       offering.provider = msg.sender;
5       if(isCategoryAllowed(category) && isPricingModelAllowed(model) &&
            keccak256(name) != keccak256("")) {
6           //store the description of the offering in a struct and save it in
                the contract memory.
7       }
8   }
```

As this function modifies the chain by adding an offering to the storage, it requires a transaction to do so. This means paying a fee in Ethers in order for this transaction to be processed. Once this is done, the offering will be on the chain. As Consumers should be able to go through each offering to assess its suitability to their product it is necessary to extract the offerings from the chain. This can be done by accessing the memory of the contract and returning an array of the informations contained in the given struct.

6.2 Delete offerings

Deleting things from a blockchain can never be done as it goes against the logic of the blockchain concept. However, it is necessary for this project to be able to delete offerings. For instance the offering could become invalid or the Provider could discover an error in the created offering. In any case removing the offering from the list is necessary. Solidity has implemented a overwrite functionality which overwrites the memory. This means that while the data inside the contract is removed, the data itself still exists in the chain. If for instance the first offering is requested deleted in the contract, it will result in the following:

ID: 1		ID: 0
Provider: 0x1c14...043		Provider: 0x0
Name: Offering 1		Name: ""
Category: TrafficCategory		Category: ""
Price: 1$ PER_ACCESS	Delete offering ⇒	Price: 0
EndTime: 30/6-2018		EndTime: 0
Endpoint: 192.168.1.101		Endpoint: ""
Inputs: Location, time		Inputs: [ ]
Outputs: TrafficDensity		Outputs: [ ]

The index in the array storage will still be there, it will just no longer have any meaningful data inside. This means any deleted offering will still take up space as the space used is not reclaimed. This will increase the search time through the offerings. A scheme to overwrite/use the holes in the storage is therefore something worth investigating as this could end up saving time as the amount of offerings grow and more offerings gets deleted. However, as the order of offerings is required to be maintained as their ID is used as a unique identifier for their location, they cannot be rearranged. This will end up leaving holes in the memory as illustrated on Figure 6.2. A simple scheme to handle this issue is to move the last element in the storage (the newest offering) into the newly deleted spot and remove the last entry in the array entirely by shortening the array size. This is much quicker than moving each element one by one to keep the integrity of the identifier intact. Such an example can be seen on Figure 6.2. However this requires remapping all

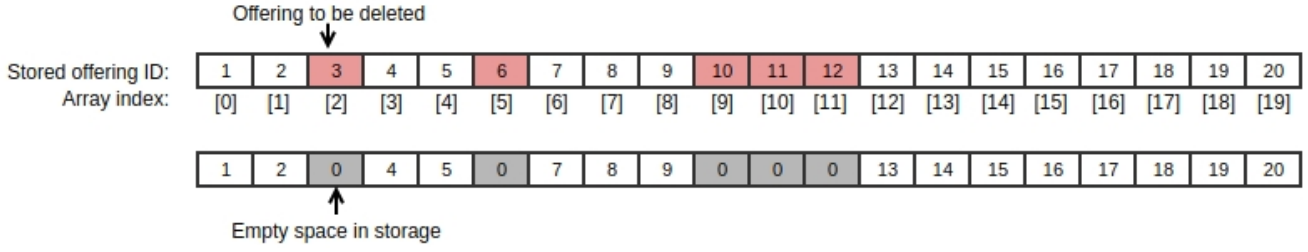


Figure 6.2: An illustration of how offering deletion looks in the offering array.

the identifiers and is not something which is wanted in the current implementation. The mapping between offerings and their locations in the current implementation relies on the ID of the offering, and these schemes

can therefore not be implemented at the current time. While this means the length of the array to search through will be larger, it is more important that the identifiers are located properly than increasing the search efficiency at this moment.

6.3 Modify offerings

Over time a provider might want to change the content of an offering. This can be done by deleting the existing offering and creating a new one. However, this would require the Consumer to subscribe to a new offering and it is therefore desirable to modify existing offerings instead. It is possible to adjust some of the different parameters of an offering. The offering parameters which can be changed are the ones which modify the offering, while keeping the offering intact. This means parameters such as inputs, outputs and endpoints can be changed as, they are specifying how and where the offering can be used. Things such as offering price and license cannot be changed as they fundamentally modifies the offering, effectively creating a new offering. Especially the price where changing the price will change the foundation of the license agreement made between Provider and Consumer. If modifications to these areas are required, the offering must instead be deleted and a new one created to replace it. To ensure that no malicious or accidental deletion or modifications of offerings are performed, the account which created the offering must also be the one to delete/modify it. If the source of the transaction does not match the offerings creator then it cannot be changed. This does mean that any offering created by a lost account cannot be removed ever. This is something worth considering.

6.4 Test of offering implementation

To ensure that the implementation functions as intended an accept test is performed. This test involves testing the implemented solutions ability to create, delete and modify offerings while also ensuring that only allowed users can do so. The tests will be performed in steps to test each functionality individually.

6.4.1 Creation of offerings

To test the creation of offerings an account calls the addOffer function in the deployed smart contract. Any successful call should result in a created offering. To test if the functionality in regards to approving pricing works, attempts with wrong parameters are also performed where they should be discarded. As it can be

Table 6.1: Tests and results from accept testing of the addOffer functionality in the offering smart contract.

Test:	Input:	Expected Output:	Actual Output:
Correct offering 1	Name: test1, Category: TrafficCategory, Pricing: FREE	Offering created	Offering created
Correct offering 2	Name: test2, Category: TrafficCategory, Pricing: PER_ACCESS, Endpoint: 10.10.10.10, Inputs: location, Outputs: trafficInfo	Offering created	Offering created
Incorrect offering 1	Name: test3, Category: TrafficCategory, Pricing: 1000, Endpoint: 10.10.10.10, Inputs: location, Outputs: trafficInfo	Failed to create offering (pricing model not allowed)	Failed to create offering
Incorrect offering 2	Name: test4, Pricing: FREE, Endpoint: 10.10.10.10, Inputs: location, Outputs: trafficInfo	Failed to create offering (missing category)	Failed to create offering
Incorrect offering 3	Category: TrafficCategory, Pricing: FREE, Endpoint: 10.10.10.10, Inputs: location, Outputs: trafficInfo	Failed to create offering (no name specified)	Failed to create offering

seen in Table 6.1 it is possible to create offerings, the offerings can be created by specifying the required inputs, however they can also be created without all inputs. As some of the settings can be changed later, such as inputs, outputs they are not required to be set when the offering is created. When trying to create offerings with invalid inputs, such as wrong pricing model, no category or no name the request is rejected. This is as intended where invalid parameters cannot be used to create a valid offering. The functionality of the addOffer implementation can therefore be deemed to perform as intended.

6.4.2 Deletion and modification of offerings

Deletion and modification of offerings requires knowing which offering to access. This means making lookups in the contract to find the offering of interest. Currently a single offering can be extracted by calling its ID. There is a function implemented in the contract which returns all offerings created by the calling address. From this list the ID of interest is then chosen. It should not be possible to delete or modify any ID not represented on this list as they do not belong to the calling address. To test these functionalities two accounts are used to create valid offerings. Account 1 will create offerings 1, 2 and 4 while account 2 will create offering 3 and 5. As is can be seen in Table 6.2 it is not possible for non-owners to modify of delete offerings they do

Table 6.2: Tests and results from accept testing the deletion and modification of offerings in the offering smart contract.

Test:	Action:	Expected Output:	Actual Ouput:
Account 1:			
Modify ID 1	Change endpoint to 192.168.1.100	new endpoint is 192.168.1.100	new endpoint is 192.168.1.100
Modify ID 2	Add inputs: Input1, Input2	Input1 and Input2 has been added to the list of inputs	Input1 and Input2 has been added to the list of inputs
Modify ID 3	Add ouputs: Output1	Denied as account 1 is not owner of offering ID 3	Denied
Account 2:			
Delete ID 1	Delete offering with ID 1	Denied as account 2 is not owner of offering ID 1	Denied
Delete ID 3	Delete offering with ID 3	All values in offering ID 3 is changed to their initial values	All values in offering ID 3 is changed to their initial values

not own. It is also possible to modify the allowed parameters. This means the desired functionality of the contract has been obtained.

Chapter 7

Offering Query Usecase Implementation

In order for a Consumer Client to be able to consume data from an Endpoint, it should first find the offering that fulfill the data requirements. Right now this is done by executing an offering query on the Consumer Client and pick one of the matching results. However, since the Consumer Client is a foreign node, it cannot search through offerings and it is therefore necessary for the Consumer to execute the offering query and return the offering to the Consumer Client. In order to not break the functionality of already implemented applications, the offering query on the Consumer Client should be the same as the one implemented in the current BIG IoT setup and then be translated into a valid Ethereum command on the Consumer. In order to perform the offering query, it is possible to create JavaScripts and load them into Geth. Geth can then call the JavaScript functions and whatever is returned from the function is then accessible. What needs to be returned from the JavaScript function are the available offerings or an error messages, the error messages could be invalid command or offerings not found. There are two possible ways of returning the offeringss:

- Return the first offering that match the criteria. This will be referred to as First Matching Result Search (FMR-Search)
- Return a list of all offerings matching the criteria. This will be referred to as All Matching Results Search (AMR-Search)

The benefit of FMR-Search is that it is faster since it does not require to search through all offerings. In Section 3.3.2 the time complexity was stated, which was set to $O(n \cdot (5 + L))$, however, for this project the search will be limited to take Category, Price and Data Field as parameters. This will give a search complexity of $O(n \cdot (2 + L))$. For FMR-Search this complexity will only be experienced if the search criteria match the last offering created or if the search criteria does not match any offerings. The drawback of FMR-Search is the unfair advantage that it gives to be the first to provide an offering with a specific set of input parameters. This problem will AMR-Search solve as the user can choose randomly of all offerings that matches the criteria and thereby avoid starvation of offerings. The drawback is that it is slower, since it needs to search through every offering created. Solidity does not support returning of tuples it is not possible to return a list of the actual offerings, but instead there can be returned a list of ID's of the offerings. The format when executing the JavaScript in the Geth terminal is seen in Code-snippet 7.1, the parameters marked with '*' are required.

Code snippet 7.1: The structure of offering query function call

```
1  // If all parameters are set
2  offeringQuery(*category, price_max, ["input_1", "input_2", ... , "input_n
       "], ["output_1", "output_2", ... , "output_n"])
3  // If only the required are set
4  offeringQuery(*category, "", [], [])
```

One thing that is important when designing the smart contract, is that it should make sure that it only use blocks of an certain age in order to avoid the inconsistency problem described in Section 2.1.2. The age calculated in this project in order to obtain five nines is 2 and was found in Section 13.3, and this protection is coded as seen in Code-snippet 7.2.

Code snippet 7.2: Inconsistency protection with age

```
1  uint start = 0; // Genesis block
```

```
2     uint end = block.number - 2; // Get current block number and extract age
3     for (uint ii = start; ii < end; ii++) {
4       // Search Code ...
5     }
```

7.1 Test of offering query implementation

In order to test the functionality four offerings has been added to the chain, seen in Table 7.1.

Table 7.1: Table containing offerings created for search test

OfferID	Name	Category	Price	Input	Output
1	Offering 1	MobileFeatureCategory	10	[1, 2, 3]	[1, 2, 3]
2	Offering 2	TransportationCategory	20	[1, 2, 3]	[1, 2, 3]
3	Offering 3	MobileFeatureCategory	30	[4, 5, 6]	[4, 5, 6]
4	Offering 4	TransportationCategory	40	[4, 5, 6]	[4, 5, 6]

7.1.1 FMR Search

On Table 7.2 seven whitebox tests of FMR-Search is shown, with their respective search criteria, expected output and the actual output. As seen in the table, the implementations of FMR-Search works as intended and it is possible to receive the desired offering IDs.

Table 7.2: Table containing results for the offering query with first match return

Search criterias	Expected Output:	Actual Output
- Category: MobileFeatureCategory	1	1
- Category: MobileFeatureCategory - Price: 10	1	1
- Category: MobileFeatureCategory - Input: [1, 2, 3] - Output: [1, 2, 3]	1	1
- Category: MobileFeatureCategory - Input: [1, 2, 3] - Output: [2, 3, 4]	No Offerings Found	No Offerings Found
- Category: MobileFeatureCategory - Input: [4, 5, 6] - Output: [4, 5, 6] - Price: 10	No Offerings Found	No Offerings Found
- Category: TransportationCategory - Price: 40	4	4

7.1.2 AMR-Search

On Table 7.3 seven whitebox tests of the AMR-Search is shown, with their respective search criteria, expected output and the actual output. As seen in the table, the implementation of AMR-Search works as intended and it is possible to receive a list of desired offering IDs.

Table 7.3: Table containing results for the offering query with all matching results returned

Search criterias	Expected Output:	Actual Output
- Category: MobileFeatureCategory	1 and 3	1, 3
- Category: MobileFeatureCategory - Price: 10	1	1
- Category: MobileFeatureCategory - Input: [1, 2, 3] - Output: [1, 2, 3]	1	1
- Category: MobileFeatureCategory - Input: [1, 2, 3] - Output: [2, 3, 4]	No Offerings Found	No Offerings Found
- Category: MobileFeatureCategory - Input: [4, 5, 6] - Output: [4, 5, 6] - Price: 10	No Offerings Found	No Offerings Found
- Category: TransportationCategory - Price: 40	4	4

Two different methods of searching was implemented, namely First Matching Result and All Matching Results. Both methods was accept tested and it was seen that both passed since the actual output was equal to the expected output. The properties of the two searching methods and how well they performs is written later in Chapter 11.2.

Chapter 8

Subscription Usecase Implementation

The general functionality of the subscription contract is close to identical to the offering contract. As with offerings it is necessary to be able to create and delete subscriptions. Unlike with the offerings, it is not possible to modify a subscription as it is based on an agreed upon license agreement. Changes to the license should and will require the creation of a new subscription to the offering. As when creating offerings, it requires a transaction to a smart contract in the blockchain to create a subscription. Making a subscription therefore also requires Ethers to accomplish.

8.1 Creating subscriptions

To create a subscription there are several checks which are required in order to ensure the subscription is valid. The functionality of creating a subscription is described by the flowchart on Figure 8.1. As the

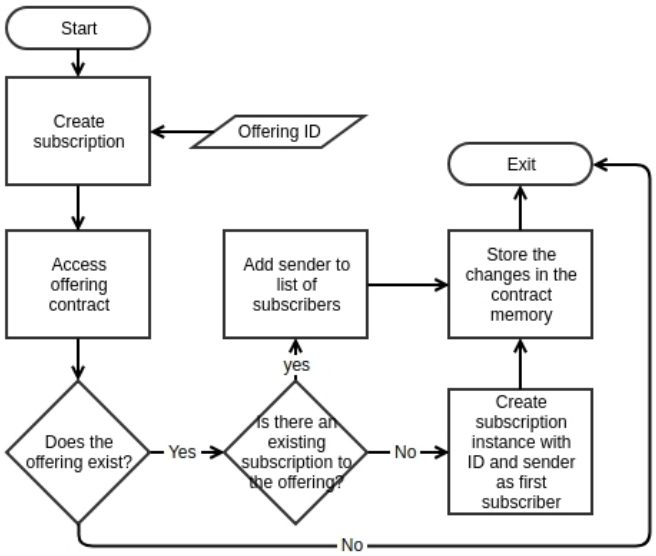


Figure 8.1: A flowchart describing the functionality of the addSubscription function in the subscription smart contract.

functionality in the subscription contract is similar to that of the offering contract it will not be explained in details. The purpose of the subscription contract is for nodes to be able to identify if a given account has a valid subscription to an offering. This is to be used to make the authentication of Consumers easier as this can now be done exclusively on the chain. Whenever a Consumer wants to create a subscription to an offering it calls the createSubscription function in the subscription contract. This function is called with the ID of the offerings, which the Consumer has found through the query method in the offering contract described in Chapter 7. With this ID number it will be registered as a subscriber to this offering. The subscriber registered is the account accessing the contract. This means no other account can delete any subscriptions made by other accounts.

8.2 Deleting subscriptions

Deleting subscriptions is a necessary feature as some subscriptions should be able to be created on a time-limited basis. Likewise a Consumer might want to cancel their subscription before time if they no longer wants access to the given Provider. Deleting smart contract entries is presented in Section 6.2. In this case, once a subscription is deleted the last element in the list is moved onto its slot to minimize the growth of the subscription list and avoiding holes. This scheme was presented in Figure 6.2 as method 2. This is possible because the order of subscriber addresses is irrelevant to the performance and functionality of the contract. This should help decreasing the search time as there are no unused entries in the list.

8.3 Test of subscription implementation

To ensure that the subscription contract functions as intended, an accept test is performed. This test will check a users ability to create and delete subscriptions. In order to create subscriptions it is required that there are existing offerings in the offering contract. This is therefore a prerequisite for this test. For this test there are three offerings in the offering contract at ID 1, 2 and 3. To create a subscription an account has to call the createSubscription function in the subscription contract with one of the three IDs. If it does not receive a valid ID the subscription should not be created. To delete a subscription the function deleteSubscription is called with the matching ID of the offering of which one wants to remove their subscription. Only the subscription creator can remove the subscription. The following tests are therefore performed as seen in Table 8.1. As can be seen on the table, the desired functionality is present. It is possible to create subscriptions to

Table 8.1: Table of tests performed to verify the functionality of the subscription smart contract.

Tests:	Input:	Expected Output:	Actual Output:
Create valid subsciption	ID: 1	True (Subscription created)	True
Create invalid subscription	ID: 4	False (offering does not exist)	False
Delete own subscription	ID: 1	True (subscription deleted)	True
Delete subscription on unsubscribed offering	ID: 2	False (no subscription to delete)	False

existing offerings, while attempts to subscribe to non-existing offerings is denied. Likewise it is possible to delete ones own subscription. This means the desired functionality of the contract is present.

Chapter 9

Access Management Usecase

As written in Chapter 3.1 both the Consumer Clients and Provider Clients are foreign nodes and should not have a copy of the chain but get the needed information through the Nodes. The Consumer Clients will be connected to a their respective Consumers and it is up to the Consumers to create authentication for their own clients. When a Consumer Client wants to access an asset from the Endpoint, the Endpoint must know if it can trust the client. In Section 3.2 there was therefore proposed to implement the authentication server as an smart contract, that could issue tokens and keep track of valid Consumer Clients. However, due to the time constrains for this project, the problem regarding issuing a token will not be addressed. This chapter will therefore only focus on how foreign nodes and chain nodes will share data.

9.1 Chain access on foreign nodes

For the foreign nodes to be able to access the chain through a node, it could be implemented with the RPC access that Geth provides. However, this solution requires that all foreign nodes to have Geth installed and this is not desired since it can require to much setup for the end user of the system. Therefore a solution without any additional packages or plug-ins needs to be found. This requires that it should be possible to make a request to a server and get all the required information from the request. To achieve this, the server must execute all the necessary Geth commands and only return the results. It is therefore required to find a server-side programming language that can execute these commands. Several solutions was investigated and the first to succeed was the PHP library web3.php. It implements the same methods for PHP as the web3.js, but it is not created by the Ethereum Group. In order to download and get it to work the following is required:

- A server that can run PHP
- PHP version >= 7.1
- mbstring extension for PHP

Now composer can be used to check if the right PHP version is installed and to install all the required packages:

```
1  // Download composer installer (composer.phar)
2  php -r "readfile('https://getcomposer.org/installer');" | php
3  // Install packages stated in composer.json and autoloader
4  sudo php composer.phar install
```

Now it is possible to get access to the chain by creating a web3 provider with an RPC address. In order to test if it is possible to access chain data on a foreign node, the server is contacted through a browser and the data that is going to be extracted are the offerings on the chain. In order to show offerings to the end users, PHP needs to get access to this contract and get all the offerings. First, PHP needs the web3 provider and the the contract by ABI and address, this is seen in Code-snippet 9.1.

Code snippet 9.1: Create Web3 Provider and Contract object

```
1  $contractABI = [ABI];
2  $contractAddress = 'address'
3  $web3Contract = new Web3('URL'); //Provider
```

```
4    $contract = new Contract($web3Contract, $contractAbi);
```

It is now possible to access the methods defined in the contract ABI, which in this case would be get offerings by their IDs. When interacting with a contract, two methods in web3.php can be used: Send and Call. Send is used when transaction needs to be made, and Call is used for read functions in the contract. Both calls need a call back function to catch any errors which are thrown. The function seen in Code-snippet 9.2 calls getOffer with the input parameter $id, if any errors are returned they are stored in $error and the offering will be stored in $result. The $result will contain an array containing the information of the offering. In order to test the functionality, the offering is printed as HTML.

Code snippet 9.2: Execution of getOffer by ID in the offering contract

```
1 $contract->at($this->contractAddress)->call("getOffer", $id, function(
    $error, $result) use ($contract) {
2   // Print Result as HTMl
3 });
```

9.2 Test of Access Management Usecase

In order to test the functionality of the foreign node access, the architecture seen in Figure 5.1 will be modified to Figure 9.1, where the Node C will be the one that contains the PHP web3 implementation and the Consumer Client will be used to extract the test results. The test results is expected to be equal to

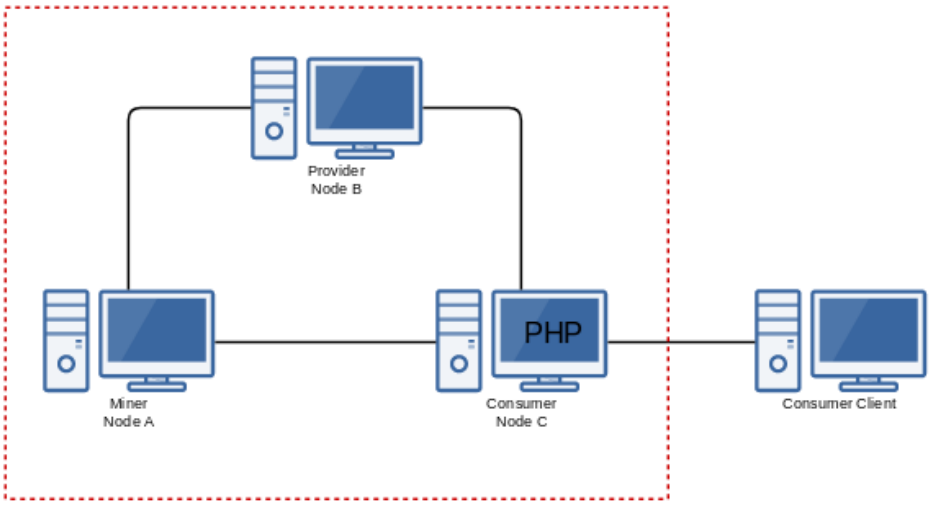


Figure 9.1: Modified architecture for the foreign node setup

the four offerings that has been added to the chain, seen in Table 9.1. The result of the code shown in

Table 9.1: Table containing offerings created for search test

Name	Category	Price	Endpoint	Input	Output
Offering1	MobileFeatureCategory	10	192.168.1.105	[1, 2, 3]	[1, 2, 3]
Offering1	TransportationCategory	20	192.168.1.105	[1, 2, 3]	[1, 2, 3]
Offering1	MobileFeatureCategory	30	192.168.1.105	[4, 5, 6]	[4, 5, 6]
Offering1	TransportationCategory	40	192.168.1.105	[4, 5, 6]	[4, 5, 6]

Code-snippet 9.2 when accessing the Node C from a the Consumer Client without any chain access is seen on Figure 9.2 and as seen all the offerings contained in the contract is shown. It is therefore concluded that it is possible to access chain data from foreign nodes.

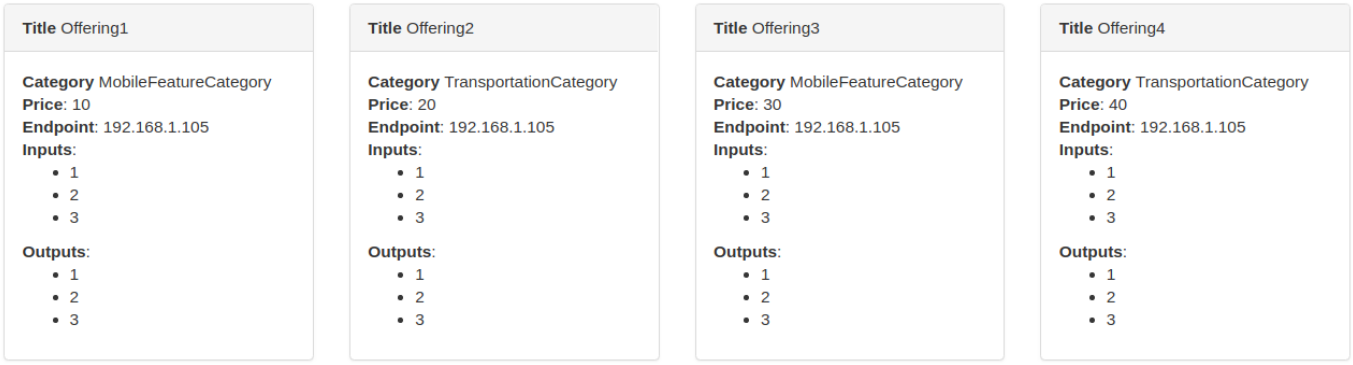


Figure 9.2: The result of a browser HTTP request from a machine without chain access

Part III

Evaluation

Chapter 10

Evaluation Areas

As described in Chapter 4 there are several areas which must be evaluated in order to conclude whether or not blockchain can provide any benefits to the BIG IoT marketplace. As written in Chapter 2 Ethereum states that scalability of the system could provide performance issues. For the marketplace, the performance issues of the system could be the search time that are affected by the amount of offerings in the chain. Another example of a scalability issue is the time it takes for all nodes in the network to observe a new offering and that could be affected by the amounts of nodes and their physical placements. So scalability in this report will therefore be evaluated throughout the different performance issues.

10.1 Time Aspects

As written, the time aspects in a blockchain system can be tested in many areas. Generally there are three different aspects where the time measures influence the performance: Mining of transactions, code execution time and searching methods.

10.1.1 Offering searching time

As different methods of implementations can impact the way a contract executes tasks it is worth looking into what performance can achieved with the contracts. Many of the tasks defined in this project requires transactions and are therefore constrained by the mining operations. This is further elaborated in Section 10.1.2. As searching is not a transaction it is not constrained in the same way. It more depends on searching through the list of offerings. This is an important part of the marketplace solution and it is necessary to evaluate. As the amount of offerings increases, so can the searching time. In Chapter 7 two solutions of searching was stated: FMR and AMR Search and both of the solutions should be tested and compared against each other. It is not only be the numbers of offerings on the chain that impact the searching time, but other factors such as number of calls to the contract and operations inside the contract can also have an impact. It is therefore worth investigating what operations are time consuming.

10.1.2 Transaction processing time

Since blocks are limited to certain amount of transactions and are created on average every 14 seconds, the system can only serve a given amount of transactions. As long as the amount of transactions does not exceed one block, any transaction broadcasted will be mined in the next block. However if there are more transactions than one block can contain, it will take multiple blocks to process. It is therefore worth looking into how many transactions the network can process, since this will indicate how the system will perform with an increasing amount of users.

10.2 Security

Security is a difficult area to evaluate as it is both large and abstract. Quantifying security analysis is difficult as security concerns can be both negligible and critical. Instead the focus is on a qualitative analysis of the security in this blockchain solution. Security in this project is therefore defined as a set of scenarios of attacks and exploits which are evaluated in terms of their effectiveness, impact and preventability. The following scenarios are going to be evaluated:

- Majority Attack
- Exploits of Smart Contracts
- Denial of Service
- Malicious Provider and Consumers
- Man in the Middle

Since this solution is based on Ethereum, it will adopt Ethereums security flaws. It is therefore important to both investigate how this solution will handle current security concerns in the BIG IoT System and the adopted security flaws.

10.3 Inconsistency

Since blockchains are decentralized there is the possibility of multiple nodes creating blocks at roughly the same time. When these blocks are distributed, some nodes will receive one block and adopt it as the newest block for their particular chain and other nodes will receive another block and use that block. This effectively creates different chains in the same network and thereby causing inconsistency on the nodes. The probability of this event occurring should therefore be determined in order to design a method to protect against this behavior.
This probability will both depend on the mining speed of the network and the block flooding time. While the mining speed is known, it is necessary to investigate the block flooding time, i.e. the time from creation of the block until all nodes has it.

10.4 Availability

Availability is the probability of the system being functional given the network size and complexity. Currently the marketplace is a centralized server and is therefore a single point of failure to the BIG IoT System. Since the marketplace functionality is moved to a blockchain it decentralizes the marketplace, and the overall availability of the marketplace should therefore increase. However, the availability on the new system depends on what perspective is taken. The marketplace is now effectively located on each blockchain node and will only experience complete failure if every single node is down. However, since this project introduced the concept of foreign nodes, it is therefore necessary to determine the availability for a single foreign node. As the availability of the individual nodes can be difficult to determine this project will instead defines the theoretical availability which can be used to calculate the availability of the system when the availability of each element is known.

Chapter 11

Time Aspects

11.1 Offering Usecase

This chapter will test and evaluate the creation, deletion and modification methods for offerings in the Ethereum private chain. The areas of interest has been described in Chapter 10. This chapter will therefore examine the rate at which different tasks can be performed, how many transaction can be done at once, any limitations to the implementation and how in general this impacts the system. Each of these will be tested in a set of scenarios customized to this particular task:

- Create offerings
  - The maximum amount of stored transactions in a blockchain node is by default 4096 in Geth [32]. This value can be increased but it is deemed unnecessary as it is unlikely for the network to experience above 4096 offering creations or subscription requests at once.
  - To create offerings a set of 4096 transactions is created to fill the TxPool of the mining node. Once this is filled the transaction are mined
- Modify offerings
  - The same limitations applies as with offerings. A maximum of 4096 offering modifications can be performed at the time
  - Because modifying offerings requires less information to be transmitted the transaction is smaller than the creation of offerings. The amount of transactions per block can therefore be more and the rate at which these can be processed is therefore expected to be bigger
- Deleting offerings
  - Deleting offerings is again a transaction and as such should scale similarly to the others. Deletion of offerings is the smallest transaction of the three and should therefore also be the one which can be processed the fastest
  - Again a maximum of 4096 deletion operations can be performed as this will fill the TxPool

The results of the performance test of the offerings is not limited to offerings only, as they test the performance of transactions in general. Therefore the results in the offering performance test will also cover the subscriptions usecase.

11.1.1 Mining transactions

As transactions are required to perform tasks in the offering contract it is necessary to test if the performance of the mining node is as expected. It is therefore important to note:

- The chain size does not directly impact transaction performance
- Internal parameters such as gas limit and difficulty does
- To ensure consistency in testing a steady state value for these parameters are used (see chapter 5.4)

This ensures that each test experiences as close to the same conditions as possible. There are three types of tasks which can be performed on offerings with the smart contract

11.1.2 Creation

The method for testing the creation offerings is the following:

1. A new, empty chain is created with a mining node and a user
2. The user migrates the smart contract created in Chapter 6 to the chain
3. The user creates 4096 offering transactions towards the smart contract function addOffer
4. Once the TxPool from the user is synchronized with the miner node, the miner node starts mining
5. The starting block ID and the last block ID is logged for later analysis

By filling up the miners TxPool it represents the scenario of 4096 offerings arriving at the same time. Instead of having 4096 nodes make one transaction each at the same time, a single node is used to create all transactions. However as this takes time mining cannot commence before the TxPool is full. Creating an offering takes an estimated 425000-525000 gas to execute, this means that given the gas limit of 8 million, a block can contain roughly 16-18 offerings. By using this we can calculate the theoretical time it takes to create offerings:

$$Time = \frac{TotalOfferings}{OfferingsPerBlock} \cdot E[BlockMiningTime] \tag{11.1}$$

As the amount of offerings will be between 16 and 18 per block depending on the parameters set in the offering, we use the average of 17 to represent the amount of offerings processed in one block for these calculations. The expectation of the tests is that the results should closely follow this calculation as the expected block mining time is adjusted constantly to be as close to 14 second average as possible [10].

$$time = \frac{TotalOfferings}{17} \cdot 14s \tag{11.2}$$

Results

To ensure that the results obtained from a test is not an outlier of the actual performance the tests are performed multiple times. This should help visualise the trend of the system. As is seen on Figure 11.1 the

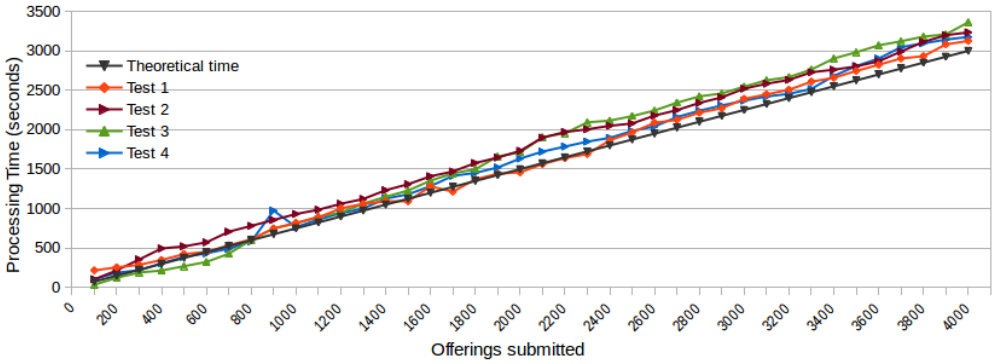


Figure 11.1: Graph showing the creation times of offerings by mining the transactions.

overall trend of the four tests are identical. They each increase linearly based on the amount of offerings to mine. This makes sense when looking at Equation 11.2. To make the picture clearer the theoretical and the average values can be seen on Figure 11.2. It is clear that the average performance of offering creation follows the theoretical calculations closely. There is almost no deviations in the comparisons between the two. The system therefore is not effected significantly when increasing the amount of offerings as the scaling is linear with the amount of offerings in the queue in front of it. However, this also speaks about the systems scalability to the extent of how it will handle an increase in offering creation rates. The results show it takes roughly 3300 seconds to process 4096 offerings. This also means that if offerings arrive faster than this it cannot process them in time, and as such its TxPool will overflow. By analysing the amount of time it takes

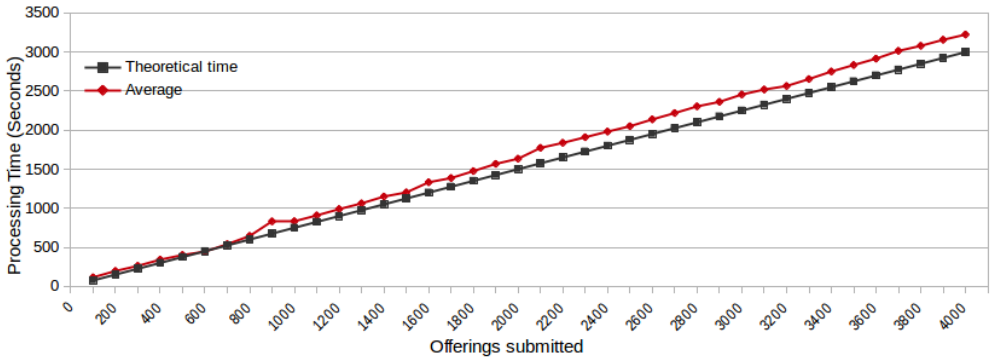


Figure 11.2: Graph showing the average creation time of offerings compared to the theoretical values calculated with Equation 11.2.

to process offerings we can calculate the rate at which offering creations must be below to avoid overflowing the miner offering transactions. To calculate this rate Equation 11.3 is used:

$$Time = \frac{NumberOfOfferings}{AverageTotalTimeSpent} \tag{11.3}$$

It is assumed that the rate of blocks mined is constant and that each block contains the same amount of offerings. This therefore looks at the time it has taken to create x amount of offerings. On Figure 11.2 it can be seen that it took 3225.25 seconds, and this results in Equation 11.4.

$$\frac{Offerings}{Time} = \frac{4096}{3224.25s} = 1.27\ Offerings/s \tag{11.4}$$

This is the maximum rate at which they can arrive on average in order to avoid overflowing the miners TxPool. If this happens the miner wont register the transaction and it would have to be resend at a later time. This can cause problems for the node who created the transaction. In order to ensure transaction order in a node, each transaction contains a node nonce. This nonce indicates the order in which transactions has been created by the node. Every time a node creates a transaction it will increase the nonce by 1. This is necessary to ensure that depending transactions are not processed in the wrong order. If a transaction is lost, the following transactions from that node can become starved until the lost transaction is retransmitted. This problem it not limited to offerings but is present in any transaction based solution.

11.1.3 Modification

The method for testing modifications of offerings is the following:

1. A new, empty chain is created with a mining node and a user
2. The user migrates the smart contract created in Chapter 6 to the chain
3. The user creates 4096 offering transactions towards the smart contract function addOffer
4. The mining node mines the offerings and stops once completed
5. The user call a modifying function in the smart contract on all 4096 offerings. In this test the deleteInput function in the contract is used. This function deletes a specified input based on string comparison
6. Once all transactions has been created and the user and miner has syncronized TxPools, the miner starts mining
7. The starting block ID and the last block ID is logged for later analysis

As modifying offerings is a smaller process than creating them, the gas it takes is lower and therefore more modifications can be processed per block. Modifications take roughly 30000 gas to complete and this allows

for 270 modifications to be processed per block. However, as they still depend on the mining of blocks it is expected that shares the linear behavior with the offering creation, but with lower values due to the increase in transactions per block. To calculate the theoretical time it would take to process the modifications we can reuse Equation 11.1 with the rate of 270 modifications per block and a 14 second expected mining time.

$$time = \frac{SumOfModificationTransactions}{270} * 14s \tag{11.5}$$

Results

As with the offering creation the first thing to look at is how the individual mined runs have performed compared to the theoretical calculations made in Equation 11.5. As we can see on Figure 11.3 they all

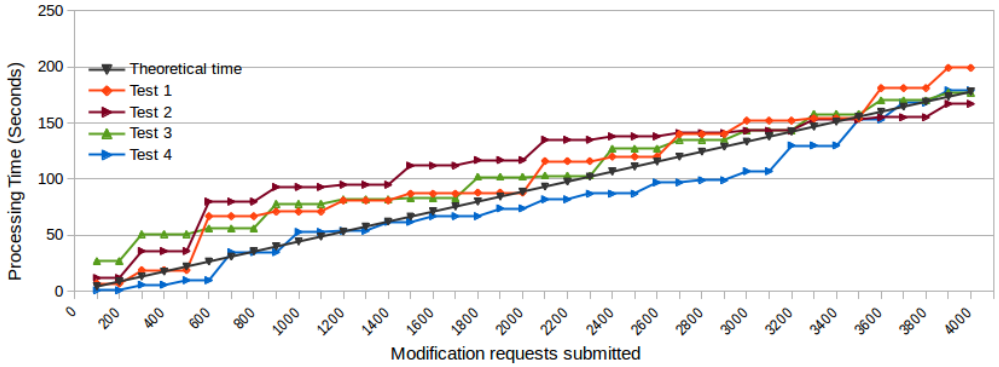


Figure 11.3: Graph showing the mining of offering modifications by mining the transactions.

share the same tendency as the theoretical calculations. However as the blocks start to contain hundreds of transactions the lines have several straight lines as all modifications in the same block has the same completion time. This is not the case with the theoretical calculations and there is therefore, as seen, small deviations. They are however close to the theoretical line and when looking at Figure 11.4 we see that the average of the runs is similar to the theoretical line. There is a higher deviation in the beginning than with the creation. As with the creation of the offerings there is a period in the beginning where the blocks has to adjust to the difficulty and gas limit of the blocks. However as there could only be 16 creation transactions in a block the amount of transactions impacted by this adjustment period was limited. In this scenario there are almost 300 transactions per block and this period is therefore much more impactful. As the amount of

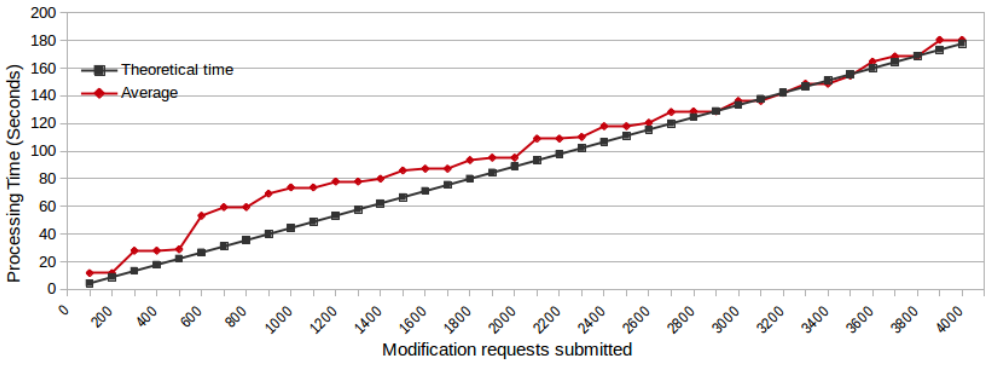


Figure 11.4: Graph showing the average of offering modifications compared to the theoretical expectation.

transactions per block with modifications is almost 20 times higher than with the offerings it is expected that the rate of which they can be processed is also higher. This is also evident by Equation 11.6.

$$\frac{Modifications}{Time} = \frac{4096}{180.5s} = 22.69\ Modifications/s \tag{11.6}$$

11.1.4 Deletion

The method for deleting offerings is the following:

1. A new, empty chain is created with a mining node and a user
2. The user migrates the smart contract created in Chapter 6 to the chain
3. The user creates 4096 offering transactions towards the smart contract function addOffer
4. The mining node mines the offerings and stops once completed
5. The user call a delete function in the smart contract on all 4096 offerings. This function deletes the specified offering
6. Once all transactions has been created and the user and miner has syncronized TxPools, the miner starts mining
7. The starting block ID and the last block ID is logged for later analysis

As deleting offerings is almost identical to modifying (modification: 30000 gas, deletion: 27000 gas) it is expected that they will share close to identical behavior. To calculate the theoretical time it would take to process the deletion of offerings we can reuse Equation 11.1 with the rate of 275 modifications per block and a 14 second expected mining time.

$$time = \frac{SumOfDeletionTransactions}{275} * 14s \tag{11.7}$$

Results

As with the offering creation the first thing to look at is how the individual mined runs have performed compared to the theoretical calculations made in Equation 11.7. As can be seen on Figure 11.5 the behavior

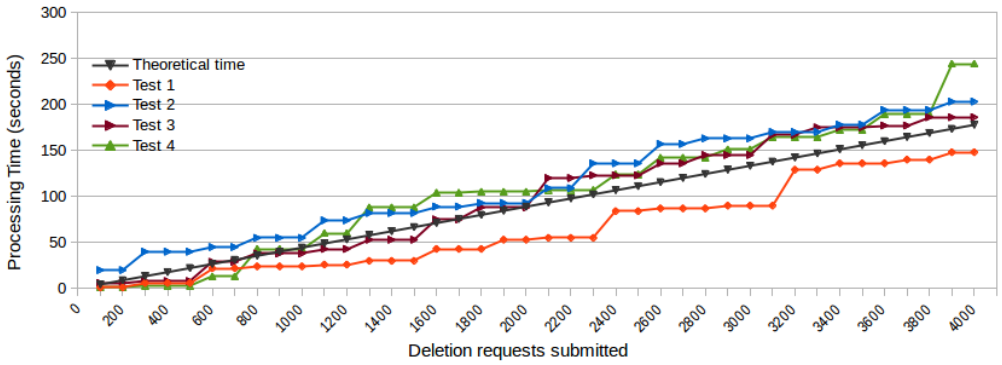


Figure 11.5: Graph showing the mining of offering deletions by mining the transactions.

of offering deletion is similar to the expected behavior. This is again seen when comparing the average of the tests to the theoretical values. This can be seen on Figure 11.6. It follows the line with only a few deviations, which can be explained by swings in mining times of blocks. When looking at the rate of which the transactions can be processed (Equation 11.8), it is also clear that the rate at which deletions are mined is much higher than that of creating offerings, however it is slightly lower than that of modifying offerings.

$$\frac{Deletions}{Time} = \frac{4096}{195.25s} = 22.06\ Deletions/s \tag{11.8}$$

This can be explained by the fact that deletion is actually a modification of every entry in the offering. As described in 1.1 deleting entries is actually not possible in a blockchain. What a deletion command does is removing all references to the data by re-initializing the values to their default setting. A deletion of an offering is therefore essentially a set of modification commands. By performing more commands at once, it takes longer to process.

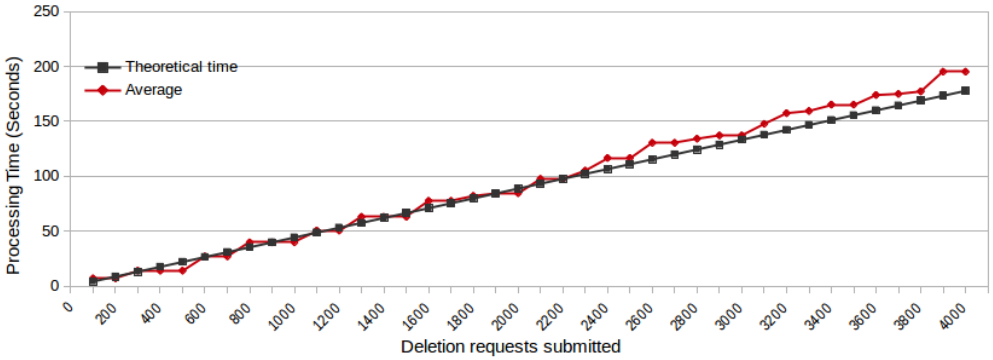


Figure 11.6: Graph showing the mining of offering deletions compared to theoretical expectation.

11.2 Offering Query Usecase

This chapter will evaluate the offering queries in the areas described in chapter 10. This includes performance of queries as well as the resulting scalability estimation. The challenge about performing offering queries is the amount of offerings. As this amount grows the time it takes to go through them will increase as well. This was lightly discussed in chapter 3.3.2 and it was seen that the search time depends on two factors, namely the amount of offerings and the amount of parameters used to describe offerings. The expectation would therefore be that offering queries scale poorly when looking at performance in regards to time both per search and total search time. While queries depends on the amount of offerings in the chain, it does not depend on other scalability factors such as nodes in the network or how the network is connected. In Chapter 7 there was stated two different searching methods and both will be tested and compared:

- **FMR:** Time to get the ID of the first matching result
- **AMR:** Time to get ID's of all matching results

There are several factors that will impact the query performance it is necessary to adjust these parameters to evaluate how querying performs in different scenarios. It is expected that it will take longer time to perform a query if the amount of offerings are large. Likewise it can be expected that querying takes longer the more specific the query is as more fields in an offering has to be evaluated.

11.2.1 FMR-Search

The following parameters are therefore adjusted to gain a more complete picture of the offering query's performance.

- Amount of offerings in the chain. This will start at 2^5 offerings and increase to 2^{12} offerings (the max value of TxPool)
- Size of the offering query. This test will compare the performance when no inputs and outputs (I/O) specified with the performance when 10 of each.
- Furthermore it will be tested how both solutions performs, if the only matching result is placed in the end of the chain

Results

The first two tests of FMR-search is seen in Table 11.1, and as seen the number of offerings on the chain does not effect the searching time when using the same searching criteria. However, it is seen that the more specific search, i.e. the more searching criteria, the longer time it takes to search since more elements needs to be validated. This is also evident in $O(n \cdot (2 \cdot L))$, as the variable $2 \cdot L$, which is the (I/O) in Table 11.1. Since $O(n \cdot (2 \cdot L))$ describes the worst case for FMR-Search, it is expected that the searching time will increase with the number of offerings (n) when no offerings match the criteria. This is also seen in Figure 11.7.

Table 11.1: FMR test with zero and ten inputs and outputs

Numbers of Offerings	Zero I/O	Ten I/O
32	81.00	101.00
64	80.00	101.33
128	81.66	104.33
256	80.66	103.66
512	81.66	102.33
1024	82.00	102.33
2048	79.66	105.33
4096	81.33	101.66

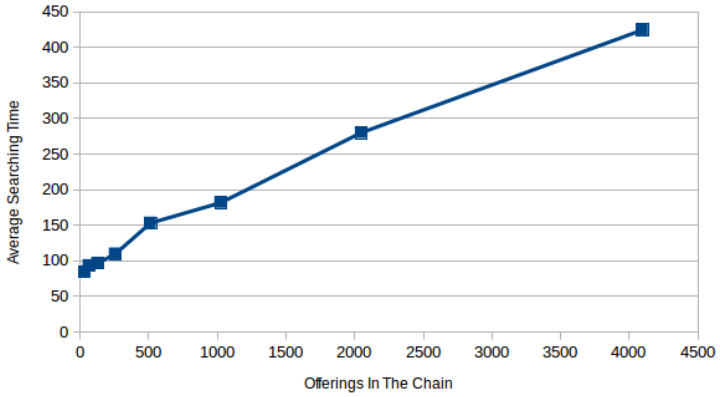


Figure 11.7: Search time of FMR search with no matching results and zero I/O specified.

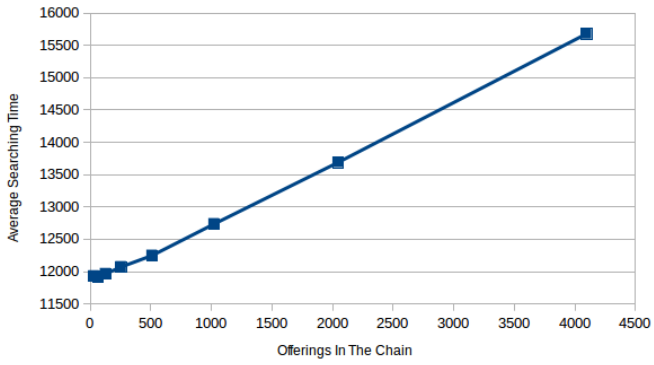
11.2.2 AMR-Search

The following parameters are therefore adjusted to gain a more complete picture of the offering query's performance.

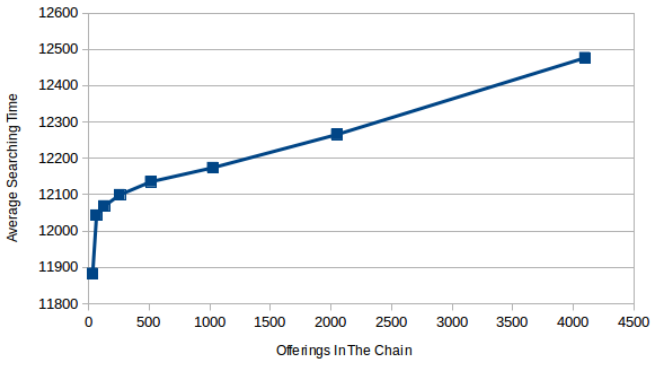
- Amount of offerings in the chain. This will start at 2^5 offerings and increase to 2^{12} offerings (the max value of TxPool)
- Size of the offering query. This test will compare the performance when no inputs and outputs (I/O) specified with the performance when 10 of each.
- Furthermore it will be tested how both solutions performs, if the only matching result is placed in the end of the chain

Results

Since AMR-Search will always loop through every offering in the chain, the expected search time will be equal to Figure 11.7. But as seen in Figure 11.8b this is not the case. So when comparing Figure 11.8b to Figure 11.7, the expected searching time for AMR-Search should be similar to FMR-Search. When looking at the searching time for 32 offerings FMR was around 90 ms compared to 11900 ms in AMR. So even though the two solutions would iterate through equal amount of offerings, AMR-Search still perform significantly worse then FMR-Search. The only difference between the FMR and AMR, when there are no matching results for the search, is that AMR creates a list (candidate list) intended for the matching results. On Figure 11.8a, the AMR-Searching time is seen when the offering query finds offerings, i.e AMR is appending to the candidate list. As seen, it does not only take time to create the candidate list, but it also takes time to append to it. In Figure 11.8b the time difference between 32 offerings and 4096 offerings is 593.66 ms, where on Figure 11.8a the time is 3748.66 ms. When comparing AMR-Search to FMR-Search, it is clear the having the searching time from FMR-Search together with the fairness of AMR-Search is desirable. Therefore a new solution of FMR-Search is created that make use of randomness.



(a)



(b)

Figure 11.8: (a) Search time of AMR search with matching results and zero I/O (b) Search time of AMR search with no matching results and zero I/O

11.2.3 Random FMR-Search

The Random FMR-Search (RFMR) takes the first matching result, but instead of always starting at the first index in the offering list, the client will send a random number between zero and numbers of offerings to the contract and use this number as an search offset. A figure of this can be seen in Figure 11.9, and shows that if the random number drawn is a zero the offering will be match 1.

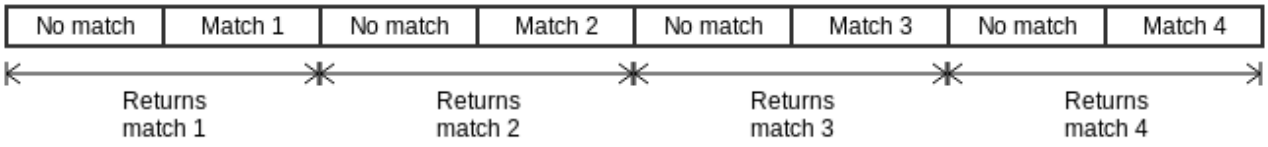


Figure 11.9: Search logic of RFMR

Test of RFMR implementation

When testing the RFMR Search on the same offerings as stated in Section 7.1 it is expected that a random of the two choices would be picked if searching only on a category. As seen in Table 11.2 it works as intended and with this solution offering with ID 3 does not starve as it would have with the FMR Search.

Table 11.2: Showing the actual output of the RFMR-Search compared to the expected

Search criterias	Expected Output:	Actual Output
- Category: MobileFeatureCategory	1 or 3	1
- Category: MobileFeatureCategory	1 or 3	1
- Category: MobileFeatureCategory	1 or 3	3
- Category: MobileFeatureCategory	1 or 3	1
- Category: MobileFeatureCategory	1 or 3	3

Results

In the performance test of RFMR two things are needed to be tested: the fairness and the searching speed. Complete fairness is defined as; all matching offerings being visited equal amount of times. So in order to measure fairness a measurement of how many hits each offering has, compared to the expected value is required. Lets define a list, X, containing the count of each hit an offering has, where n is the number of

matching offerings:

$$X = x_1, x_2, ..., x_n \tag{11.9}$$

For the searching algorithm to be completely fair, each offering needs to be picked equally amount of times, which is expected to be:

$$\bar{x} = \frac{m}{n} \tag{11.10}$$

where m is the number of searches. Instead of defining fairness, this project will define unfairness and is in this project described as the root mean square error of the difference between the expected amount of picked compared to the actual:

$$unfairness = \sqrt{\frac{\sum_{i=0}^{n}(x_i - \bar{x})^2}{m}} \tag{11.11}$$

The more unfair the search is, the higher the value will be. When having the same matching distribution as seen in Figure 11.9 it is expected that RFMR would have approximately the same searching time as FMR and have a low unfairness value. The searching time is seen in Table 11.3 and is close to the FMR searching times seen in Table 11.1. The unfairness in Table 11.3 is created with 2000 searches (m=2000). The theoretical max value will depend on the number of searches and will occur when one of the offerings receive all hits. The theoretical boundaries for the unfairness values can also be seen in Table 11.3 in the column: Unfairness boundary. The worst-case scenario of RFMR is equal to FMRs if the last offering is the only match, as

Table 11.3: FMR test with zero and ten inputs and outputs

Numbers of Offerings	Zero I/O	Ten I/O	Unfairness	Unfairness boundary
32	77.00	96.33	1.06	[0, 44.04]
64	79.67	95.33	0.80	[0, 44.38]
128	79.00	97.00	1.08	[0, 44.58]
256	82.67	96.67	0.80	[0, 44.63]
512	74.33	95.33	0.96	[0, 44.68]
1024	81.67	94.67	0.94	[0, 44.70]
2048	88.67	96.67	1.00	[0, 44.71]
4096	80.67	101.00	1.00	[0, 44.72]

RFMR would need to loop through every offering in the chain to find a match. This is seen on Figure 11.10. In the implementation test it was seen that neither of the MobileFeatureCategory results starved, but this

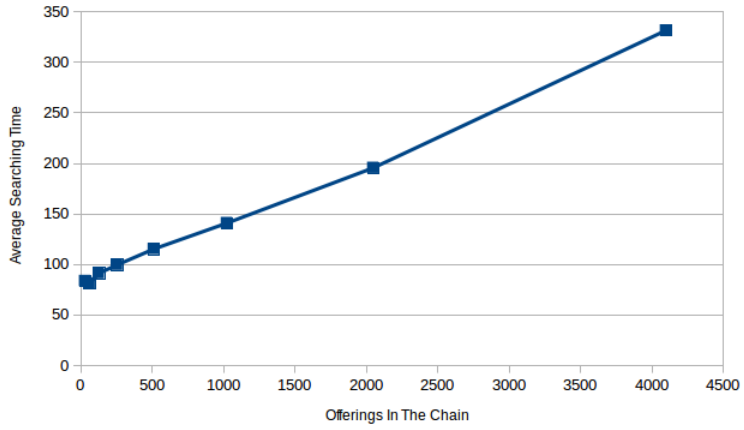
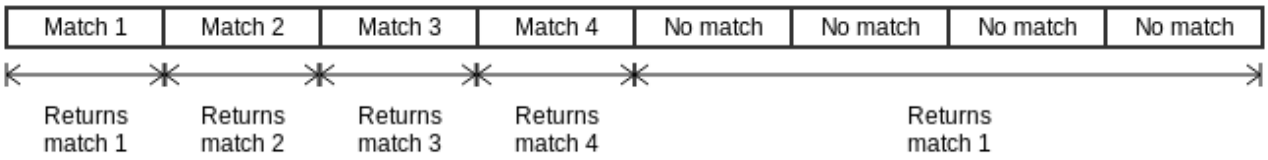
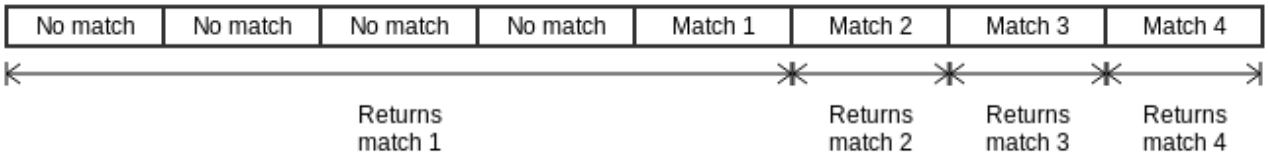


Figure 11.10: Search time of RFMR-Search with no matching results and zero I/O

does not necessarily mean that starvation of offerings cannot occur or that every offering will be visited equal amount of times. In the previous example of RFMR illustrated in Figure 11.9 all offerings had the exact same amount of random numbers that can pick it, but in the real case the matching result placement can be more or less random. Therefore the worst-case can be seen on Figure 11.11a and Figure 11.11b and in both cases the first offering will still have an unfair advantages. In Table 11.4 three runs with 2000 RFMR



(a)



(b)

Figure 11.11: Offering order that leads to unfair RFMR-Search

searches with an matching offerings distribution equal to the example in Figure 11.11a where the first 256 offerings are the matching offerings out of a total 1024 offerings. In the first column shows the number of hits for the first offering and the middle column is a summation of numbers of hits for all other matching offerings. The last column shows the unfairness value for every run. When drawing a random number in $X \sim U(0-1023)$, every number above 255 will result in the offering with index zero, and it should therefore give a high unfairness value. The expected amount of hits for the first index will be equal to Equation 11.12.

$$\frac{n-(n'-1)}{n} \cdot m \tag{11.12}$$

where n (1024) is the number of all offerings, m is the number of searches (2000) and n' is the number of matching offerings (256). In this unfair search, it is expected that the first matching result will get:

$$\frac{1024-(256-1)}{n} \cdot 2000 = 1502hits \tag{11.13}$$

As seen 11.4, even though the starting point is random, RFMR can still be unfair due to the distribution of

Table 11.4: Showing the potential unfairness of RFMR with clustered matching offerings

First Matching Offering	All Other Matching Offerings	Unfairness Value
1496	504	33.35
1508	492	33.61
1485	515	33.10

the matching offerings.

11.2.4 Summary

As all tasks related to creating, modifying and deleting offerings are implemented as transactions its performance has been tested. Is has been seen that the general trend for the transactions processing time is linear which is expected as they rely on the mining operations of the Ethereum blockchain. As the transactions has different sizes depending on what task it represents the rate at which they can be processed varies. The offering creation is the largest transaction and the system can therefore only process 16-18 of these transactions per block. The modifications and deletion transactions are smaller and almost 300 of these can be in each block.

As the offering query methods does not require transactions, the searching time is not depend on mining. However, several things can be extracted from these tests. It was expected that FMR-Search and

AMR-Search, when the offering query did not match any offerings, would have the same searching time, described as $O(n \cdot (2 + L))$. But this was not the case as the result was that AMR-Search performed significantly worse than FMR-Search. The only difference between the implementations is that AMR-Search stored all matching results in a list, and it can therefore be concluded that memory operations in a smart contracts is time consuming. This result should be remembered whenever creating a smart contract in the future, since it will affect the performance of the contract.
Since FMR-Search introduced a significant unfairness flaw, RFMR was created. It was desirable to have the searching time FMR but the possibility of fairness of AMR. RFMR would randomly pick an offset from where to start the search and this helped avoid returning the first matching offering. However, RFMR worst-case could be almost as unfair as normal FMR.

Chapter 12

Security Concerns

In Chapter 3.1 a new combined architecture for BIG IoT on blockchain was created. This introduced elements which exits in a normal blockchain architecture, while also introducing a new concept of foreign nodes that does not have access to the chain. This means that all security concerns from the blockchain technology itself is adopted in this solution and it is therefore important to clarify what those are. Furthermore, the idea of having users outside the chain could in itself provide security concerns, and it is therefore also important to clarify what new security concerns the combined architecture will introduce. This chapter will therefore look into the general blockchain security concerns and the project specific security concerns.

12.1 General Blockchain Security Concerns

Throughout this report it has been investigated how the BIG IoT system can be implemented using Ethereum Blockchain. While the concept of mining adds security in the form of trust, Ethereum still have some issues. It is therefore important to uncover the general blockchain security issues that are important to remember when using the blockchain technology. This chapter will therefore introduce problems that the blockchain technology can have, and translate it into the implementations that have been done it this report.

12.1.1 The 51% problem

Since blockchains relies on decentralized verification of blocks, it is up to the individual users to help validate the blocks that are discovered in the network. The most prominent problem in this areas is called the 51% problem and occur when a miner has above 50% of the total mining power. If is happens the miner will be able to mine a majority of the blocks and in such an event they are able to perform denial-of-service (DOS) attacks on the blockchain. This is done by denying certain transactions or address from being mined. It is furthermore possible for the miner with the majority of mining power to double-spend Ethers as the attacker secretly can mine on a private chain, that will grow faster than the network. Since the age mentioned in Section 13.2 is used to determine when a transaction is valid, the attacker has to build a private chain longer than the age from the point of the transaction, and then introduce the private chain to the network. It is important for the private chain to also be longer than the networks chain, otherwise it will be rejected.

In the context of this project the DOS-attack could prevent a user from creating offerings, gaining authorization or create subscriptions. For the double spending issue, it will be possible for the attacker to receive a token, access the data on the Endpoint and then revert the transaction. Thereby not having spend any Ethers in doing so.

12.1.2 The Blockchain Anomaly

The blockchain anomaly, presented in [23] is a problem closely linked together with the 51% attack, but have some differences. The attack can occur when a user can manipulate the delay between mining nodes, as it can force forks longer than the age threshold. The idea is to split the network into two chains and then make transactions on the loosing network. This would allow the attacker to make double spending transactions on the winning chain, and thereby gaining the same benefits as an 51% attack. While this sounds like a problem, the reality is that it can be hard, if not impossible to force delays between nodes in a network. However, this problem can occur naturally in a network, if the spatial spread of the nodes are large and the

mining rate is too fast. This is also one of the reasons why Ethereum has chosen the mining rate to be 14 seconds, in order to lower the forking probability. and thereby also decreasing the risk of this event happening.

As the solution in the project has not changed the mining rate, the blockchain anomaly is not seen as a possible attack. However, it indicates that the mining difficulty in the genesis block most be sufficiently high, in order to avoid a fast mining rate in the beginning.

12.1.3 Solidity Problems

In [13] several Ethereum contract security techniques and tips are stated. The problems in Solidity is not necessarily something that will be exploited by a malicious user, but is something that can occur unintended by an unaware developer. However, some malicious user can create smart contracts for malicious purposes.

DOS with infinity Loop

As all transactions are processed even if they are valid or not, it is possible to create transactions which are not valid but still needs to be processed. So what happens if a smart contract has a function, which only contains an infinity loop? The miners that are validating this transaction will get stuck in this transaction and the transaction attack is therefore a DOS attack. However this problem has been addressed and solved by Ethereum, by the introduction of gas. As stated in Chapter 2 gas is payment for processing in the miner nodes. This means that any maker of a transaction has to pay gas for it to be processed. If a infinity loop is introduced in the transaction, the transaction will eventually run out of gas. The attacker would therefore be required to have large amounts of Ethers to spend in order to perform the DoS attack on the blockchain.

On Ethereums public chain, Ethers is valuable and therefore a transaction attack would be extremely expensive. However in the context of this project, Ethers does not present any actual worth, other than the ability to perform transactions. This means, that if a miner in a network possess a lot of Ethers, the miner can perform transaction attack for as long as the funds allows. If the miner successfully mines new blocks while performing a transaction attack, the period of the attack would also increase, and if this attack is performed by a node with more that 51% of mining power, this attack can be extended for a large amount of time.

DOS with Block Gas Limit

In order to limit the blocksize, Ethereum has introduced block gas limit, which limit the amount of transactions inside a block. This introduces a security concern with contracts that takes an unknown size of input parameters into the function call. For every parameters that the function take in, the gas size of the transaction will increase and this might increase above the gas limit and this will block the transaction from being completed.

This is something that can occur in the implementation of the add offering functionality presented in Chapter 6. Here the Provider can create an offering with as many input and output parameters as he likes. When looking at gas fee table in Appendix A it is seen that it costs 68 gas for every non-zero byte of data or code for a transaction. As the block gas limit is set to $8 \cdot 10^6$ this problem will most likely not be an issue in this implementation. However it is still unwise to not follow the coding convention from Ethereum, which suggest to count the gas used by a function. If the function call would exceed the block gas limit, the function should throw an error.

12.1.4 Transparency

An important part of the blockchain concept is the transparency in transactions. Any transaction made can be viewed by all parties. This is necessary in order for miners and other nodes in the P2P network to validate balances of accounts as well as allow transactions between users. In this project the transparency ensures that any node is able to validate a transaction as well as access the smart contracts. However it also means that anything is public. Any user has full access to the content and structure of the smart contracts.

Any potential vulnerability in the contracts are then visible to potential malicious users. Another problem could turn out to be the fact that other users can see each others transactions. This would allow a company to determine what subscriptions a competitor has and thereby create a similar or better service. This is not a problem which can be mitigated as long as the blockchain technology is used. However this is not deemed a severe issue by this project.

12.2 General Project Security Concerns

12.2.1 Malicious Consumer and Provider

As there is no functionality in this project regarding who is allowed to create providers and consumers, this can become an issue in regards to unwanted users. As long as they have the genesis file it is possible to become a member of the BIG IoT system. It is then possible for the new Provider to create fake offerings which would either point towards a non-existing or malicious Endpoint or return wrong or malicious data. If a Consumer in good faith subscribes to this offering the malicious Provider could receive money for a service which is not provided. As an extension of this it would be possible for a Provider to take advantage of free offerings and create identical offerings with the obtained data which requires payment. While is not directly a malicious attack, this is highly undesirable, since it both cheats the original Provider and the Consumer that pays for data that was intended to be free. Another possible issue is that the Provider can create a lot of different offerings with different parameters and a lot input and output parameters. This will give a higher chance that the RFMR algorithm from Chapter 11.2 will return one of fake offerings.

12.2.2 Handling of tokens in Access Management

One security concern in the Access Management is the handling of tokens. As seen in the sequence diagram in Figure 3.3 there is not stated any encryption of the token. This means that the token will be sent as plain text from the Consumer to the Consumer Client and everyone that will listen to this transaction will be able to obtain the token (lets call this a man in the middle an abuser). With the token, it is possible for the abuser to access the data without paying and depending on the license this can be on behalf of the Consumer. However, as seen in Figure 3.4 it requires more than just the token to access the data, since it also requires the Consumer address, Endpoint address and the right data inputs. So for an abuser with no knowledge of the BIG IoT system it can be difficult to actually make use of the token, however for abusers with knowledge of how the system works it can be done. This could be malicious Providers that want to steal data from their competitors. One way to prevent sniffing of tokens would be to encrypt the token. This could be done with PKI encryption, where the authentication contract will encrypt the token with the Consumer Clients public key and to make use of the token, it requires the corresponding private key. Right now, the Consumer Clients does not have any key pairs and this needs to be further investigated.

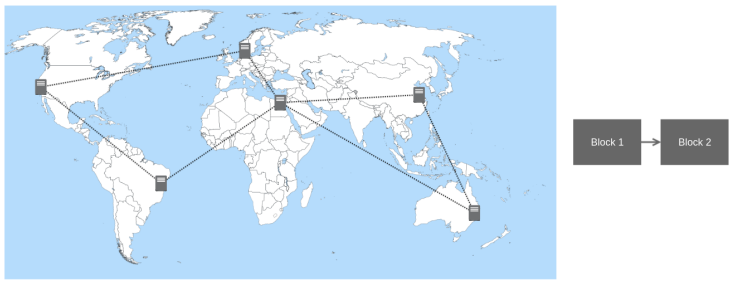
Chapter 13

Inconsistency

As written earlier in this report the blockchain technology can experience the lost update problem (see Chapter 2). In this context the problem is called forking and this chapter will try to determine a model which can be used for inconsistency protection.

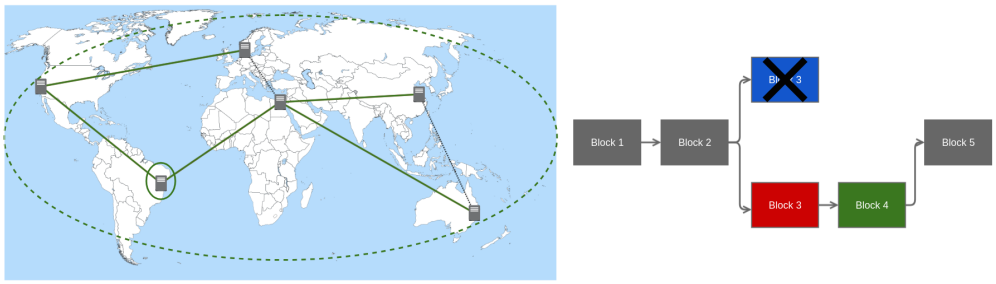
13.1 Forking Problem

An example of forking can be seen on Figure 13.1. In Figure 13.1a all nodes are working on the same



(a)

(b)



(c)

Figure 13.1: (a) All nodes have the same chain. (b) Two separate nodes each mine a block almost simultaneously and there are now two versions of the chain in the network. (c) A node with the red chain finds a new block (green), thereby eliminating the other versions of the chain.

blockchain. In the Figure 13.1b, the two marked nodes each mines a block almost simultaneously. One node

mines the block marked in red, another mines a different block marked in blue. They each distribute the block to their respective peers and the block propagates through the network. However, at some point in the network the two blocks will intersect. In such a situation the first arriving block will be chosen as the correct block. This results in two separate blockchains being active, and neither one can be classified wrong as they simply represent different views of the chain. This state will remain until the fork is resolved. A fork is resolved by simply choosing the chain with the most accumulated amount of work (i.e. the path with the most combined difficulty). Such an event is seen on Figure 13.1c where the marked node mines a new block on top of the red chain. This extents the amount of work (difficulty) used to create this chain and it is now seen as a better alternative to the blue chain. All nodes will therefore switch to this new green chain. This effectively eliminates the blue chain by overwriting the blue block with the red-green pair of blocks. The fork could have been extended if the blue chain had mined another block before receiving the green chain. In the event of a fork, any blocks contained in the losing path is discarded.

Some transactions will have to be mined again on the new path as they were only contained in the discarded blocks and this can lead to some problems. The possibilities of forks introduces an interesting problem in the form of when a transaction can be deemed committed. Ensuring that a block stays in the chain can therefore mean waiting until multiple blocks has been mined on-top of it, as this lowers the chances of the blocks being replaced. Forks occur more frequently at lower difficulties as it becomes easier for nodes to mine new blocks. By increasing the difficulty forks becomes more rare but will slow down the rate at which it is possible to mine transactions. This problem can also lead to double spending of Ethers if Bob spends his newly earned Ethers before the losing chain collapses. Bob might not do this on purpose, but there exists several methods of exploiting this blockchain flaw and this is described in Chapter 12.

13.2 Forking Probability Model

In this project, the problem of forking means that consumers in general can subscribe to an offering that in the future will be removed due to forks and the subscriber will, in theory, be able to double spend Ethers. Likewise the subscription transaction towards the offering could be lost, thereby not granting the user access to the data. The goal is therefore to create a scheme which, based on the probability of blockchain forking, will wait for x amount of blocks before a transaction can be used. This scheme will then be used in the offering query implementation. It is therefore important to clarify how old blocks should be before they are seen as stable. This problem can be split into two sub problems:

- What is the change of a fork occurring at a given block instance?
- What is the probability that the fork lasts x blocks? This will be called age of the fork

Both questions will be answered under the following assumptions:

- All miners have equal mining power, i.e. the block mining times are i.i.d
- All miners are working on the same block number
- The difficulty has converged, i.e. the rate of mined blocks in the entire network is on average 14 seconds.

Lets define the probability p as a miner who finds a block and successfully distribute it while no other node in the network discovers a block in the distribution period, i.e no forks. This can be seen on Figure 13.2a, where T_{TR} is the time it takes to distribute the block to all nodes in the network. The probability 1-p must then be the probability of a fork. This can be seen on Figure 13.2b, where T_M is the mining time of the second block for that block number in the network. The probability of no forking is therefore the probability that T_M is larger than T_{TR}, see Equation 13.1.

$$P(NoFork) = P(T_M > T_{TR}) \tag{13.1}$$

It was stated in [27] that the time distribution of block discovery is seen and proven to be exponential, giving us the PDF seen in Equation 13.2:

$$f(t)_X = \alpha \cdot e^{-\alpha \cdot t} \tag{13.2}$$

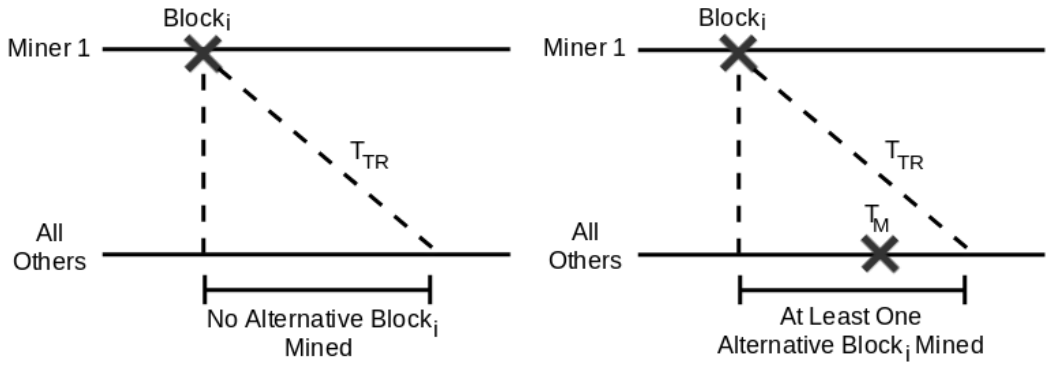


Figure 13.2: (a) Case where no forks occurring in the network (b) Case where at least one fork occurring in the network

where $\alpha = \frac{1}{14}$ is the mining rate (one block every 14 seconds on average), t is the time spend and X is the distribution time of a block discovery, which is a collection of each individually miners distribution time. Mining is memoryless, which means the amount of tries does not affect the chance of finding a block. This mean when Miner 1 finds a block and begins to distribute it, the probability of any other node finding a new block is equal to the CDF of Equation 13.2 at time t. The probability of no fork can therefore be described as:

$$P(NoFork) = 1 - CDF = 1 - (1 - e^{-\alpha \cdot t}) = e^{-\alpha \cdot t} \tag{13.3}$$

However, this is only valid when T_{TR} is constant, and this might not be a fair assumptions. Therefore it is needed to investigate how the distribution times are distributed in a P2P network.

13.2.1 Network Delay

In order to calculate the probability of creating blockchain forks in the network it is required to determine the time it takes to distribute a block. This time will determine the chances for other mining nodes to create their own block. The distribution time of packets in a blockchain P2P network is impacted by multiple factors. Amongst those are the network topology, network size and spatial diversity. If the nodes are fully connected the block flooding time will be equal to the highest delay between the mining node and the receivers. However, as the topology is not necessarily fully connected the flooding time will depend on which node mined the block. The more spacial diversity the nodes experience in the network (physical/infrastructural distance between nodes) the longer time it can take to transmit blocks. In order to keep this delay estimation within scope of the project some areas will be examined while others will be determined by simple assumptions.

Network Topology

The topology of a P2P network can vary a lot and can be difficult to determine as nodes can create and destroy paths in the network at runtime. In Ethereum the P2P network is created by utilizing the bootnodes described in Section 5.2 to create and maintain the P2P connections. As nodes join the network they will contact the bootnodes and get peers addresses. The optimal situation is a fully connected network, however as this peer discovery is slow there are times where the network is not fully connected. This section will define three scenarios of network topology:

- Fully connected network
- Mesh network
- Line network

With peer discovery the network will eventually reach a steady-state topology of fully connected, see Figure 13.3a. This is the best case scenario. If all nodes are connected to each other the network delay will be as low

as possible. However, as peer discovery takes time, the topology of the network will until this steady-state be a mesh network, see Figure 13.3b. The old nodes in the network will likely be fully connected to each other but the new nodes are not included. The worst possible topology is a line topology, here the nodes will only have a single connection to its neighbor, see Figure 13.3c.

(a) (b) (c)

Figure 13.3: (a) Fully connected network which is the ideal scenario. (b) Mesh network, the state of the network until steady-state. (c) The worst possible topology network delay wise.

The delays for distributing a packet in these scenarios is vastly different. Each link in these topologies represents the time it takes to distribute a packet from point A to point B. If the network is fully connected the distribution time is the link between the node who mined the block and the receiving node. The longest distribution time is therefore the longest link. In the line topology the distribution time is the sum of links from the mining node to the other nodes. the distribution time in mesh networks can be quite difficult as links between nodes can vary depending on the mesh type. Therefore this project will focus only on a fully connected network.

End-to-end Delay

To properly determine the distribution time of a block in the network the values of the links in the topology must be found. However the problem here is that network delay is impacted by things such as network load/-conditions which vary from scenario to scenario. In [35], [31] and [30] it was determined that the exponential distribution is the best fit for modelling network delay. It was concluded that the exponential distribution is a better fit than a truncated normal distribution, but it was in [29] shown that it could benefit from combining the exponential and truncated normal with a weight factor. [17] presents a Gamma distribution for modeling network delay based on recursive values. While the community is not in total agreement it points to the fact that the exponential distribution is a viable choice. This will therefore be the chosen distribution for network delay in this report.

Since the distribution time of a block is assumed to an exponential distribution, the probability of forking will be equal to Equation 13.4.

$$\text{Given } X \sim Exp(\mu) \text{ and } Y \sim Exp(\lambda), \text{ then: } P(X > Y) = \frac{\lambda}{\lambda + \mu} \tag{13.4}$$

This represents the probability of a single fork event occurring at any given block number. Next, it is desired to know the probability of a fork reaching a certain age. The age can be described as a Markov Chain, see Figure 13.4, where every state represents the age, starting from age zero where no fork is present. The probability of changing state is the probability of an fork occurring (1-p), since a fork does not depend on previous events. The probability of a fork being revolved must be the probability of no fork (p) at any state. No matter how many different versions of the chain exist in a network, the case in Figure 13.2a will resolve all forks. Therefore any state can return to state zero (no fork) with probability p.

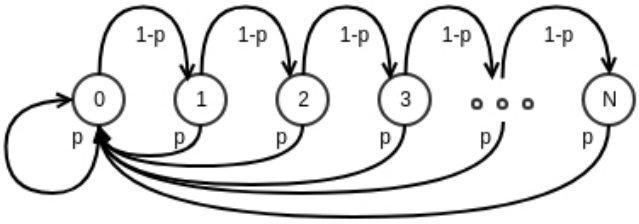


Figure 13.4: Showing the age as a Markov chain with transit state probabilities

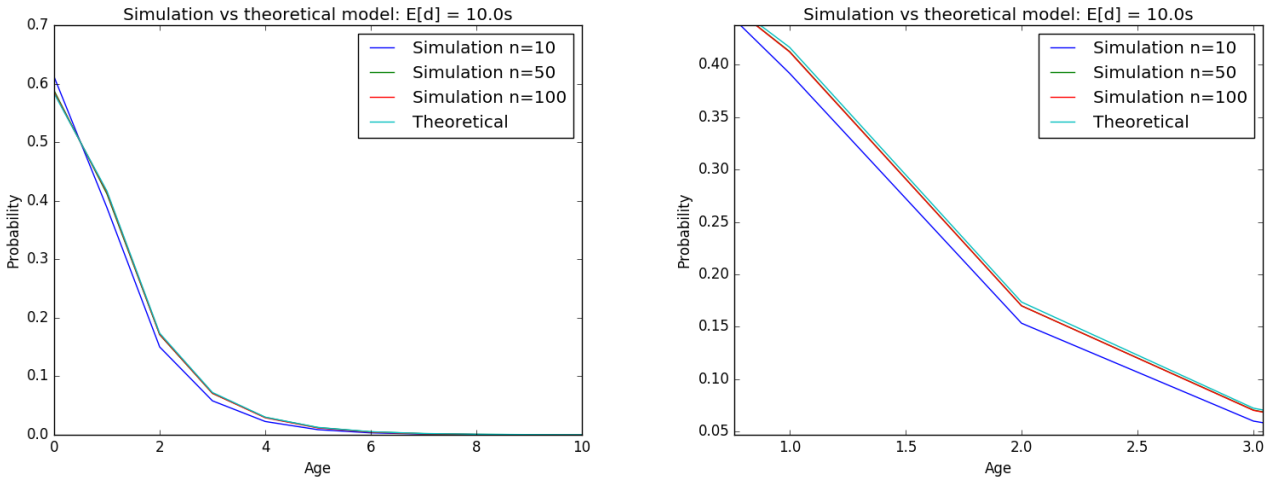
13.3 Simulation

In order to verify the forking probability shown in Equation 13.4, the forking scenario is simulated. The simulation picks n exponentially distributed random mining times, where n is the number of miners, and sorts the mining times. Likewise an exponentially distributed random variable d represents the distribution time for the block. The two lowest mining times extracted and compared to check if the time between them is larger than the distribution time d. If the result is lower than the distribution time, the system will fork, see Code-snippet 13.1.

Code snippet 13.1: Simulation of forking

```
1 blockDiscoveryTimes.sort()
2 if((blockDiscoveryTimes[1] - blockDiscoveryTimes[0]) < d):
3   forks = forks + 1
```

10000 iterations of this process is performed and the probability of a fork reaching a certain age is calculated. There are two parameters which will impact the simulation, namely the amount of nodes and the distribution time. First, the amount of nodes in the network is compared to the model. For this simulation the distribution time will have a mean of 10 seconds. This simulation can be seen on Figure 13.5a. As the lines are close together, a zoomed version of the same graph is seen on Figure 13.5b.



(a) (b)

Figure 13.5: (a) The probability of a forked block age given exponentially distributed flooding time with rate 1/10 and varying amount of mining nodes. (b) Zoomed version of the same graph

As can be seen on the figure there are small deviations between the simulations depending on the amount of nodes. As the node amount increases it gets closer and closer to the model. This suggests that the model is most accurate in large networks. While the difference is negligible the simulations will from this point be performed with 100 nodes in the network. As the block distribution time has a high impact on this probability several runs is performed with different values. It is important to note that the simulation takes the same

assumptions described in Section 13.2 as the model. Figure 13.6 shows the simulation results compared to the model derived in this project, where 100 nodes attempt to mine a block and the distribution delay has rate $\lambda = 1/0.2$ i.e. 200ms distribution time. As can be seen the simulation is close to identical to the model.

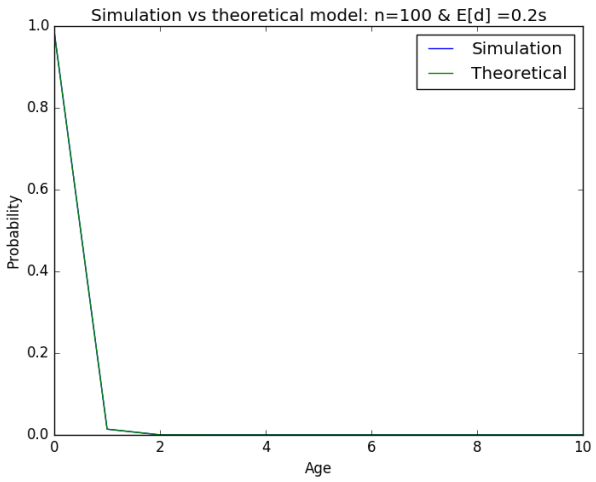
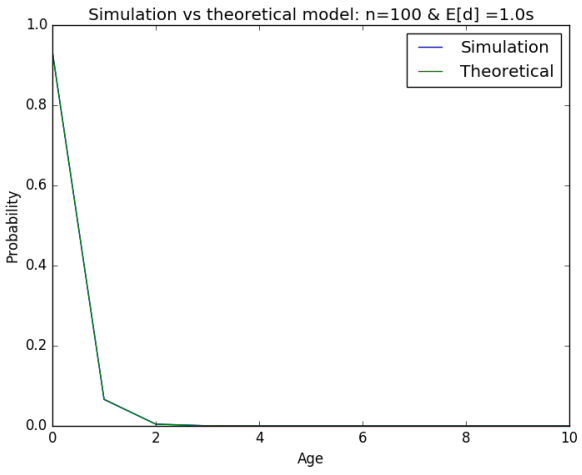
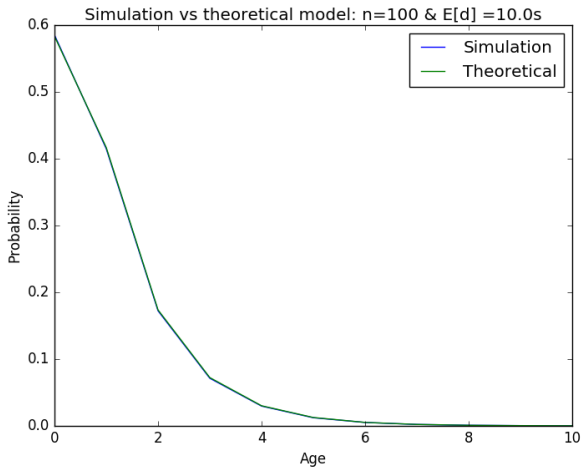


Figure 13.6: Probability of a forked block age given exponentially distributed flooding time with rate 1/0.2 and 100 mining nodes.

There is a small deviation between the lines which is difficult to notice without zooming in. However, this deviation is minute and it is expected that it is a result of inaccuracy of rounding inside the simulation. When increasing the distribution time the simulation still follows the model. It is furthermore seen that, as expected, the probability of forking is increased when increasing the expected distribution time from 1 second (see Figure 13.7a to 10 seconds (see Figure 13.7b). An interesting thing occurs as the distribution



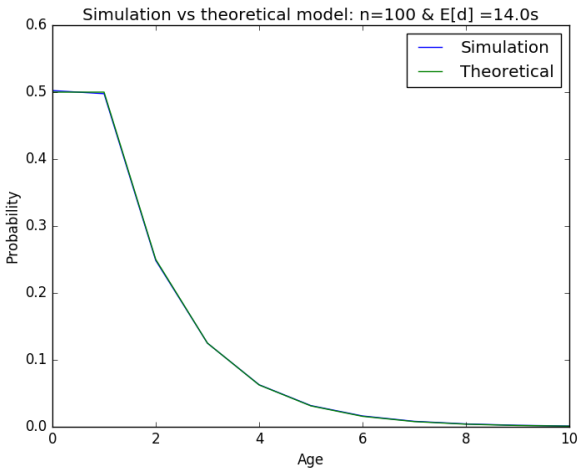
(a)



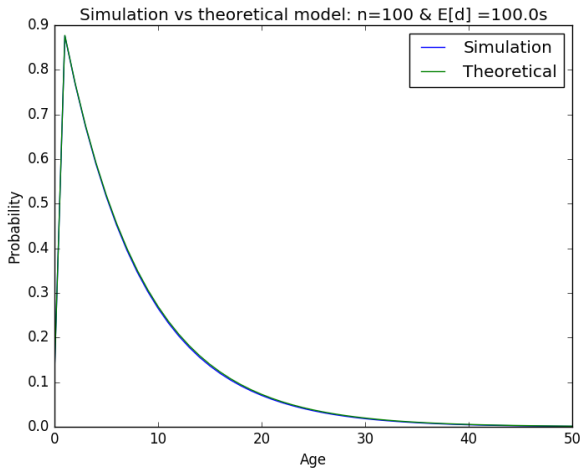
(b)

Figure 13.7: (a) Probability of a forked block age given a mean distribution time of 1 second and 100 mining nodes. (b) Probability of a forked block age given a mean distribution time of 10 second and 100 mining nodes.

time increases. The probability of the block being forked at least once becomes higher than its probability of being not forked. On Figure 13.8a it can be seen what happens when the distribution time is equal to the block mining time of 14 seconds. Here the probability of the p event will be equal to the 1-p event. After this point, the probability of an 1-p event will be larger than the p event, this is seen on Figure 13.8b with 100 seconds distribution time, note that Figure 13.8b shows age up to 50. The system becomes more and more unstable as the distribution time of blocks increases. When looking at Figure 13.8b it is clear that once



(a)



(b)

Figure 13.8: (a) Probability of a forked block age given an exponential distribution time with a mean of 14 second and 100 mining nodes. (b) Probability of a forked block age given an exponential distribution time with a mean of 100 seconds and 100 mining nodes.

the distribution time is much higher than the expected block generation time, forking becomes increasingly problematic. As the model fits the simulated results it is now possible to use the model to calculate the exact amount of blocks this project has to wait before blocks can be used. In order to determine this value a real-world representation of the network delay is required. In order obtain this, several ping tests are performed towards providers currently existing in the BIG IoT Marketplace [15]. It should be noted that as ping returns only round trip times values these are halved to represent one-way delay. This might not be a true representation of the actual one-way delay, but is deemed valid for testing purposes.

- **gibo.fib.upc.edu:** Located in Spain. Delay: 33.44ms
- **nviot.netvalue.eu:** Located in Italy. Delay: 47.07ms
- **bigiot.lab.es.aau.dk:** Located in Denmark. Delay: 0.39ms
- **big-iot.nissatech.com:** Located in Serbia. Delay: 26.86ms

For testing purposes the worst case value is selected as the average rate at which a block can reach its destination. By using this parameter we can again calculate the probability of a fork reaching a certain age, this is seen on Figure 13.9. The probability of a fork occurring is 0.0033 while the probability of reaching a fork of age two is 0.000012 according to our model. In order to achieve the coveted five nines it is therefore not necessary to wait for more than 1 block, i.e. the second block can be used. However, if the size of the network, and especially the physical distance between nodes, increases it would be necessary to re-evaluate this scheme. Table 13.1 shows a few examples of what the first usable block is depending on the mean delay of the worst link in a fully connected network. As the calculations in this report suggests, it is only neces-

Table 13.1: Usable blocks given the average delay of the worst connection in the P2P network.

Delay mean	50ms	100ms	250ms	500ms	1s	5s	10s	20s	100s
First usable block	2	3	3	4	5	9	14	22	88

sary for the network specified in this report to wait 1 block. This raises the question: Why does Ethereum recommend waiting 11 blocks?. The reason for this is can be found in 1) The assumptions made in Section 13.2 and 2) the fact that the model only focuses on accidental forking.

The assumptions made in Section 13.2 have an impact in regards to the results determined in this chapter.

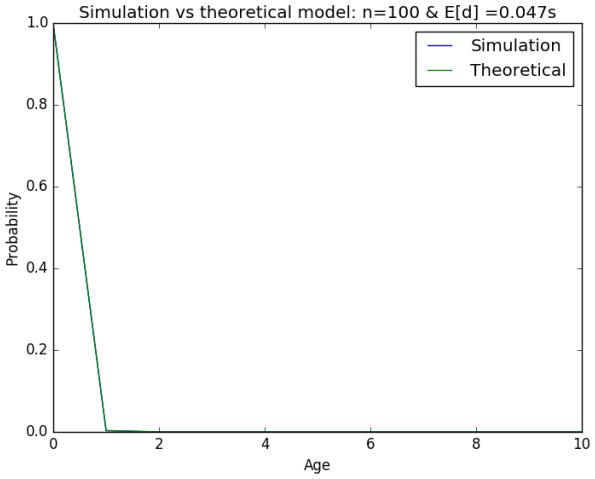


Figure 13.9: Probability of a forked block age given exponentially distributed flooding time with the acquired ping value and 100 mining nodes.

Firstly it is assumed that the miners all have equal mining power. This is rarely the case unless dedicated devices are proclaimed mining nodes. If nodes are not of equal mining power then the scaling of the difficulty will vary depending on which node is added to the network. More importantly this also means that some nodes have higher probability of discovering a block as they can perform more calculations per second than their counterpart. In such an event the model will be inaccurate since the most powerful node was the one to discover the first block. If it was, then the probability of forking will be lower than the model predict. If it was not, then it has a higher probability of discovering a block in the distribution time, and the probability of a fork is larger than the model. Finally it is assumed that all miners are mining on the same block. This is an extension from before where, as some nodes have yet to receive the newest block, they are not in contention to mine the next block after that. As such the amount of miners for a given block depends on the speed at which the previous block was distributed. This actually decreases the chances of a fork as fewer nodes are able to mine the block of interest.
While these assumptions are relevant in regards to the final result, the real reason why Ethereum implementations typically use 11 blocks [24] is because forking is not always accidental. While accidental forking occurs randomly in the network they are often resolved quickly as shown in the simulations. However, intentional forking made by attackers is another problem entirely. In [26] the creator of Ethereum, Vitalik Buterin, discusses how attackers can manipulate chains several blocks in the past. These types of attacks are further described in Chapter 12 but the main point is that the more mining power an attacker can acquire the further back in the past they can change transactions. This problem reaches its peak when the attacker possesses 51% or more of the mining power. In such an event any block has the potential to be changed. In order to take this into account it is necessary to determine how difficult it would be for an attacker to acquire a large chunk of the mining power. If for instance an attacker controls 25% of all mining power their probability of them being able to double-spend is the probability of them being able to mine enough blocks in a row to exceed the age threshold. With the current 1 block waiting scheme the probability of them being able to double-spend is equal to the probability of mining two blocks in a row:

$$P(TwoConsecutiveBlocksMined) = 0.25^2 = 0.0625 \tag{13.5}$$

By possessing 25% of all mining power the attacker would be able to create a transaction, mine it, have it validated and then retract it with a probability probability of 0.0625. In order to combat this issue a larger age threshold should be used as Ethereum suggests when taking malicious nodes into account.

13.4 Summary

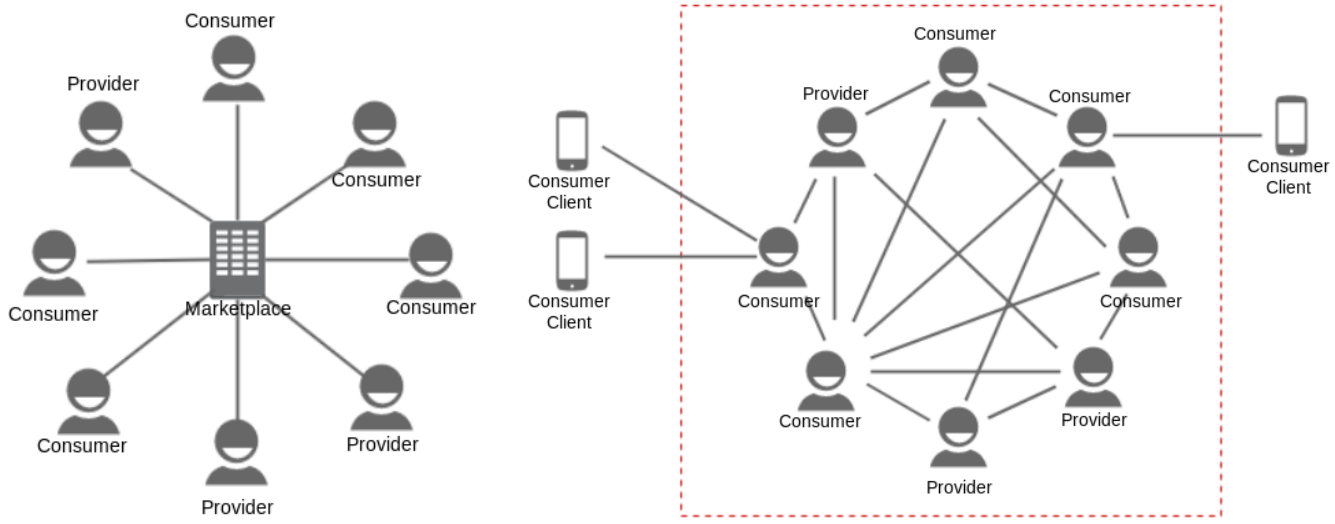
The motivation behind determining the blockchain forking probability was to be able to design solutions for the BIG IoT system that does not get influenced by inconsistency in the network. In order to determine

this probability two distributions were needed: The mining rate distribution and the block distribution time distribution. For the mining rate, the distribution was seen to be exponential with an rate of $\alpha = \frac{1}{14}$. For the distribution time the distribution was also seen as exponential, however the distribution rate is depending on a number of factors, e.g. on the network typology, the spatial diversity and connection type. It was therefore not possible to determine a distribution rate that covers all aspects of the block distribution and some rates was therefore measured and used. The model for calculating the forking probability was based on a few assumptions and some of the assumptions does have an impact on the result. First assumption was that every node in the network has the same mining power, i.e. the block mining rate for every miner is i.i.d. However, as seen in both Ethereum Homestead and Bitcoin miners trying to increase their chances of finding blocks faster than their competitors by increasing their mining power. In this case, the probability of forking is lower, since the miner that finds the first block could have the majority of the mining power and thereby leaving smaller probability for the rest of the miners to find a new block in the distribution time. The second assumption was that every miner are working on the same block number which in some few examples will not be the case. If some miners are working on older blocks than the newest distributed one, the alpha of the block discovery would decrease and the probability of a fork will lower. These two assumptions means the model presented in this report is a worst-case model and it is therefore a safe model to base the design solutions upon.

Chapter 14

Availability

Right now, the current architecture is centralized with the BIG IoT marketplace as the central unit, this is seen on Figure 14.1a. This means that Consumers and Providers will not be able to perform any actions if the marketplace goes down. As blockchain provides a decentralized solution, the availability should increase in the solution as single point of failure is removed, see Figure 13.8b. The actual availability of this new



(a) (b)

Figure 14.1: (a) The current architecture of the BIG IoT marketplace. (b) The proposed architecture of the BIG IoT on Ethereum Blockchain

architecture is difficult to determine as no values for the individual node availability is known. However a theoretical calculation can be made and it will depend on which unit that is in focus:

Consumer: If the Consumer goes down, it is not possible for the Consumer Client to make subscriptions, queries or get authenticated, which is required to obtain a token. The probability of a Consumer being available will be denoted as $P_C = P(NodeBeingOnline)$. there is no reason to calculate the probability of every Consumer node being available, as there is no existing functionality that allows Consumer Clients to borrow other Consumers for chain access. Such a functionality would however greatly increase the availability of the system as it would work as long as a single node is online.

Provider: If the Provider goes down, it is not possible for the providers to make offerings on the chain. Furthermore it is not possible for the Provider Endpoint to authenticate tokens received by Consumer Clients. The availability of a Provider will be denoted as P_P. Again, the Endpoint can only use its own Provider to authenticate tokens, and therefore a collective availability is of no use. It is not possible to use other Providers as there is no shared trust between them which makes it difficult to trust an authentication made by another node.

Foreign nodes: From the Consumer Clients point of view, the availability is seen as the the probability of successfully receiving the desired data. This requires its Consumer Client to be available in order to

make the offering query, get subscribed to the offering and receives the access token. To get the data with the token the Endpoint must be available, this will be denoted as P_E. Last but not least it requires that the Provider is up, for the Endpoint can authenticate the token. The collected availability for the Consumer Client to collect the data will then be: $P_{CC} = P_C \cdot P_P \cdot P_E$. If the Consumer Client already has to token for the data and does not need to perform the query the necessity of P_C disappears.

14.1 Example Availability Calculation

An example of the availability of the new architecture can be calculated by assuming each node is independent and has an availability of 0.99999 also known as five nines. The probability of a Consumer or Provider node being available is therefore $P_C = P_P = 0.99999$ This is the same as in the current system where the marketplace would then have an availability of five nines which is equal to roughly five minutes down time per year. When looking at the foreign nodes probability of retrieving data, it is seen that:

- Without a pre-existing token: $P_{CC} = 0.99999 \cdot 0.99999 \cdot 0.99999 = 0.99997$
- With a pre-existing token: $P_{CC} = 0.99999 \cdot 0.99999 = 0.99998$

In the current BIG IoT system the same scenario is translated into the marketplace being available as well as the endpoint on which the data should be gathered and the authentication server. This results in an availability of 0.99997. As the example shows the availability has not gotten better. This is because the amount of nodes required in order to acquire data is the same. While the authentication server has been moved to the chain a Consumer or Provider node is now required. However, if the marketplace goes down in the current system it will affect all Consumer and Provider, while in the proposed architecture a breakdown on a Consumer or Provider will only affect a small part of the system. So even though it availability is not changed from the end users point of view, the overall availability should have increased. Furthermore, if a scheme of utilizing other Consumer nodes is implemented this will help increase the system availability.

Part IV

Final Remarks

Chapter 15

Conclusion

One of the main goals of this report has been to implement the BIG IoT usecases presented in Chapter 3. As the BIG IoT system is not directly compatible with the Ethereum blockchain technology a new combined architecture has been created. The BIG IoT actors of Consumers and Providers are implemented as blockchain nodes and most of the BIG IoT functionality is created as smart contracts. This allows every node access to the same code and the BIG IoT developers can have full control over allowed functionality. The transparency that exists in blockchain technology makes it important to design the contracts with a method for disabling its functionality. It is furthermore important to make any critical variable private and to make the contract as generic as possible in order to avoid manipulation.

As forks can occur in the network it is important that consumers does not use an new offering instantly. It is necessary to specify a waiting period before blocks are used. This waiting period has been defined as block age and is integrated into the design of the offering query. Age has in this report been defined as the probability of accidental forks reaching m blocks with probability of less than 0.00001.

As Solidity is a relatively new programming language, it lacks some features. One feature is the ability to return tuples and it has therefore been necessary for the smart contracts to perform the search internally. It was observed that some operations in Solidity are extremely time consuming compared to others, these were operations which store data inside the smart contract such as arrays. The final design therefore avoided these actions in order to improve the execution time of the contract.

As the implementations in this report utilizes the Ethereum blockchain they will adopt both the advantages and disadvantages of Ethereum, of which some has been discussed in this report. Since every node in the network have the same chain it removes some elements of single point of failures, thereby increases the availability. It furthermore also ensures some aspects of security, such as integrity, as it becomes difficult to manipulate blocks. Since blocks are linked together with hashing it becomes difficult to change the content of one block as every block hash after that would need to be recalculated. One of the disadvantages that blockchain brings is the time it takes from a transaction is made until it can be seen on the chain. On average it takes 14 seconds for a new block to be discovered, and together with the forking problem as mentioned above, the expected time it takes for a transaction to become visible on the chain would be $14s \cdot ageThreshold$. If the BIG IoT in the future is required to handle real time functionality it would be close to impossible to achieve this with the blockchain technology. Further issues arise in the ever increasing blockchain size. Whenever a block is discovered it is added to the chain, and if there is no transactions in the TxPool an empty block will still be mined. This will lead to an ever increasing block chain size and it is therefore required for the users of the BIG IoT system to have enough space to the chain. Even as it has already been decided the chain should not be on the Consumer client this problem could still return as time passes and the storage requirement for the Consumer and Provider nodes increases. It is therefore necessary to consider the trade-off between mining speed and chain growth for an implementation with is required to run continuously.

While Ethereum can add positive elements to the system it also comes with a cost, so whether Ethereum will be a good technology or not highly depends on the system. This project has shown that it is possible to achieve the necessary functionality of the BIG IoT Exhange part on the Ethereum platform, however there are several trade-offs between performance, security, availability as well as development and operational requirements for users and BIG IoT themselves.

Chapter 16

Future Work

16.1 Ethers and Mining

No matter if the chain is public or private, Ether has to be used for transactions in the network. On the public chain the currency can be achieved by mining in the network or by purchasing the currency from other users. In the public chain Ethers is a currency, which in it self hold a value and can be bought with regulated currencies. This means that the motivation behind mining in the public chain is to gain Ethers. However, the value of Ethers and the motivation behind mining can be more of less absent in the private chain. However, Ethereum builds the functionality upon Ethers and they are still necessary in order to do operations on the chain. When a provider wants to interact with the chain, i.e. create, modify or delete an offering, it requires Ethers to do so. It is therefore necessary for the accounts to have sufficient Ether to perform these operations. As the users can be expected to create multiple offerings or subscriptions it is necessary for them to have access to sufficient funds. So the true value of Ethers in the presented solution is the ability to make offerings and subscriptions. As these operations cost little Ethers, there can be little motivation from nodes with lot of Ethers to keep mining. As Ethers also provides security aspects in Ethereum, where one of them is removing infinity looping codes, they are a necessity. Therefore it is not an option to remove Ethers from the system, and it is therefore required to both find a suitable value for Ethers while keeping the motivation for mining.

16.1.1 Mining

The core concept of blockchains is that mining blocks ensures the consistency and integrity of the data in the chain. If there are no miners the system will not function. Even if BIG IoT decides that they will mine it is relatively easy to perform the 51% attack on the chain as the existing mining power will be limited. Future work could therefore be to investigate the possibilities of forcing Consumers and Providers to mine, by modifying the Geth implementation. While this can work there must be a way to validate that the nodes are using the modified Geth version. There are currently no method of checking whether or not a node is mining and as such it is difficult to exclude a user who is not mining. Further investigations therefore has to be made into the methods of gaining miners in the network in order to secure the functionality and integrity of the implementation. Several blockchains are looking into switching from Proof-of-Work mining to a popular concept called Proof-of-Stake. Proof-of-Stake is a method of creating blocks where mining is not required. However the concept has yet to be fully materialized and a working version for Ethereum is yet to be developed. If this switch is made in the future the performance in regards to mining related operations can increase dramatically as block no longer requires time-consuming mining to be formed.

One element that has not been considered in this report is the varying gas price a Consumer or Provider can set on the transactions. If the number of transactions in the TxPool exceeds the capacity of one block, it is not necessarily the oldest transactions that would be mined first. The miners will always try in increase their profit when mining and it would most likely the most valuable transactions that is mined first. One future work will there be to investigate, when transactions should be marked as legacy, in order to avoid starvation. The gas price it self could also be investigated, as it might make sense to make it constant, thereby making the TxPool close to a FIFO queue.

16.2 Solidity Optimization

Blockchain as a technology is still in its infant stage and there is therefore much still left to be investigated, evaluated and improved. The technology has been increasingly popular in the past few years but has yet to be picked up by the scientific community as a viable research area. As such there is little research done in regards areas such as performance and security as well as what types of implementation can be achieved. Currently there is no state of the art worth mentioning in this report as all publicly available information regarding blockchains are the currency aspects as well as hobby implementations. The decisions made in regards to design and implementation can therefore quickly become outdated as the technology evolves. This is especially evident in regards to the use of Solidity. As Solidity is still under development new features arrive constantly, many of the features known i current programming languages are still missing and this has had consequences for the implementations. Once newer, smarter, features arrives in Solidity it would be worth updating the current implementations. This has especially been evident in the query implementation where it has been impossible to return tuples in the current version of Solidity. This has meant several modifications had to be made in order to achieve the desired functionality and this could have an impact on the performance of ths system. Revisiting the implementations once a new Solidity version is released is therefore recommended.

16.3 Availability Optimization

In order for the Consumer Client to access the data, it needs to obtain an access token from its Consumer. When obtained, it can use this token to access the Endpoint, which will validate the the token at its Provider. This means that the availability, seen from the Consumer Clients point of view, depends on both its Consumer, Endpoint and Provider. In order to increase the availability, one future work could be to investigate if Consumer Clients could use other nodes than its own Consumer to obtain access tokens. The same could be investigated for the Endpoint, however both raises the question of how the mapping could be done and if the foreign nodes can trust the chain node and vice versa.

16.4 Emulation Of Forking Model

The project has attempted to emulate the results gained in chapter 13 but has run into several problems in this regard. Mining on a device utilizes all the allocated CPU power it gets, which per default is all of it. It therefore attempts to use the entire CPU power of the device. As more and more miners are run on the same device, the amount of total allocated CPU power stays the same. The CPU consumption is not the only problem this project has experienced. Another issue has been to find a method of achieving exponentially distributed delay between devices in the testbed. As all devices run Ubuntu the easiest choice would be to utilize its netem package which allows for emulating different types of networking functionalities. However netem does not have an implementation of Exponential delay and this had to be developed by this project in order to achieve the desired properties. As such it is necessary to do emulation on a setup capable of running a large, fully connected, P2P mining network, in order to fully verify the model derived in this project.

Bibliography

[1] Dubai 10X. Dubai 10x initiative presents world's 1st vehicle database. http://dubai10x.ae/dubai-10x-initiative-presents-worlds-1st-vehicle-database/, 2018.

[2] Sebastian Peyrott Developer at Auth0. An introduction to ethereum and smart contracts. https://auth0.com/blog/an-introduction-to-ethereum-and-smart-contracts/, 3 2017.

[3] Martin Lorenz (Atos), Klaus Cinibulk (Atos), Wolfgang Schwarzott (Atos), Arne Broering (Siemens), and Achille Zappa (NUIG). Deliverable 4.1b: Design of marketplace. http://big-iot.eu/wp-content/uploads/2016/04/D4.1b-Design_of_Marketplace.pdf, 9 2017.

[4] BitInfoCharts.com. Cryptocurrency statistics. https://bitinfocharts.com/, 6 2018.

[5] Rajiv Cheriyan. Let's get started with your first ethereum dapp! https://medium.com/@rajiv.cheriyan/lets-get-started-with-your-first-ethereum-dapp-f09feb59dd78, June 2017. Figure showing the P2P network of a Ethereum blockchain.

[6] Ethereum community. The homestead release. http://www.ethdocs.org/en/latest/introduction/the-homestead-release.html, 2016.

[7] Intel Corporation. A guide to the internet of things infographic. https://www.intel.com/content/www/us/en/internet-of-things/infographics/guide-to-iot.html.

[8] State Of The Dapps. Featured dapp collections. https://www.stateOfTheDapps.com, 2018.

[9] Etherchain.org. Evolution of the average block gas limit. https://www.etherchain.org/charts/blockGasLimit. Gas limit over time in Ethereum homestead public blockchain.

[10] Ethereum. Mining. https://github.com/ethereum/wiki/wiki/Mining, May 2018. Ethereum documentation of mining.

[11] The Ethereum Foundation. Ethereum blockchain platform: Build unstoppable applications. https://www.ethereum.org/.

[12] Dr. Gavin Wood Co. founder of Etherum. Ethereum: A secure decentralised generalised transaction ledger. 2018.

[13] James Ray Ethereum Group. Safety. ethereum contract security techniques and tips. https://github.com/ethereum/wiki/wiki/Safety#ethereum-contract-security-techniques-and-tips, 6 2018.

[14] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, 1991.

[15] BIG IoT. All offerings. https://market.big-iot.org/allOfferings, 5 2018.

[16] David Siegel Kryptodesign.com. Understanding the dao attack. https://www.coindesk.com/understanding-dao-hack-journalists/, 6 2016.

[17] Murk Kulmun and Bernd Information Systems Laboratory Stanford University Girod. Modeling the delays of successively-transmitted internet packets. 2004.

[18] Alex Leverington. Rlpx: Cryptographic network & transport protocol. https://github.com/ethereum/devp2p/blob/master/rlpx.md#node-discovery, March 2018. Specifications for the peer-to-peer protocols used by Ethereum.

[19] Maersk. Maersk and ibm to form joint venture applying blockchain to improve global trade and digitise supply chains. https://www.maersk.com/press/press-release-archive/maersk-and-ibm-to-form-joint-venture, Jan 2018.

[20] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. *Peer-To-Peer Systems: First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002*, 2002.

[21] Microsoft. Peer-to-peer - transactional replication. https://docs.microsoft.com/en-us/sql/relational-databases/replication/transactional/peer-to-peer-transactional-replication?view=sql-server-2017, August 2016. Microsoft SQL server 2017 documentation.

[22] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[23] Christopher Natoli and Vincent Gramoli. The blockchain anomaly. *CoRR*, abs/1605.05438, 2016.

[24] Christopher Natoli and Vincent Gramoli. The balance attack or why forkable blockchains are ill-suited for consortium. pages 579–590, 2017.

[25] Vitalik Buterin Creator of Ethereum. Ethereum scalability and decentralization updates. https://blog.ethereum.org/2014/02/18/ethereum-scalability-and-decentralization-updates/, Febuary 2014.

[26] Vitalik Buterin Creator of Ethereum. On slow and fast block times. https://blog.ethereum.org/2015/09/14/on-slow-and-fast-block-times/, 9 2015.

[27] A. Pinar Ozisik, George Bissias, and Brian Neil Levine. Estimation of miner hash rates and consensus on blockchains (draft). *CoRR*, abs/1707.00082, 2017.

[28] Community Research and Development Information Service (CORDIS) European comission. Big iot - bridging the interoperability gap of the internet of things. https://cordis.europa.eu/project/rcn/200833_en.html, 6 2017.

[29] E. S. Sagatov, D. V. Samoilova, A. M. Sukhov, and N. I. Vinogradov. Composite distribution for one-way packet delay in the global network. pages 1–4, Nov 2016.

[30] Andrei M. Sukhov and N. Yu. Kuznetsova. What type of distribution for packet delay in a global network should be used in the control theory? *CoRR*, abs/0907.4468, 2009.

[31] Andrei M. Sukhov, N. Yu. Kuznetsova, A. K. Pervitsky, and Aleksey A. Galtsev. Generating function for network delay. *CoRR*, abs/1003.0190, 2010.

[32] Péter Team leader at Ethereum Szilágyi. Command line options. https://github.com/ethereum/go-ethereum/wiki/Command-Line-Options, November 2017. Explaination of commands available in Ethereums CLI.

[33] Truffle. Running migrations. http://truffleframework.com/docs/getting_started/migrations, 2018.

[34] P. Ward and G. Dafoulas. *Database Management Systems*. Fasttrack Series. Thomson, 2006. pages 182-196.

[35] Y. Xia and D. Tse. Inference of link delay in communication networks. *IEEE Journal on Selected Areas in Communications*, 24(12):2235–2248, Dec 2006.

[36] Vlad Zamfir. Introducing casper "the friendly ghost". https://blog.ethereum.org/2015/08/01/introducing-casper-friendly-ghost/, august 2015.

[37] Zastrin. Ethereum architecture. https://www.zastrin.com/courses/1/lessons/2-3, 2018.

Appendix

Appendix A

Gas fee table [12]

Name	Value	Description*
G_{zero}	0	Nothing paid for operations of the set W_{zero}.
G_{base}	2	Amount of gas to pay for operations of the set W_{base}.
$G_{verylow}$	3	Amount of gas to pay for operations of the set $W_{verylow}$.
G_{low}	5	Amount of gas to pay for operations of the set W_{low}.
G_{mid}	8	Amount of gas to pay for operations of the set W_{mid}.
G_{high}	10	Amount of gas to pay for operations of the set W_{high}.
$G_{extcode}$	700	Amount of gas to pay for operations of the set $W_{extcode}$.
$G_{balance}$	400	Amount of gas to pay for a BALANCE operation.
G_{sload}	200	Paid for a SLOAD operation.
$G_{jumpdest}$	1	Paid for a JUMPDEST operation.
G_{sset}	20000	Paid for an SSTORE operation when the storage value is set to non-zero from zero.
G_{sreset}	5000	Paid for an SSTORE operation when the storage value's zeroness remains unchanged or is set to zero.
R_{sclear}	15000	Refund given (added into refund counter) when the storage value is set to zero from non-zero.
$R_{suicide}$	24000	Refund given (added into refund counter) for suiciding an account.
$G_{suicide}$	5000	Amount of gas to pay for a SUICIDE operation.
G_{create}	32000	Paid for a CREATE operation.
$G_{codedeposit}$	200	Paid per byte for a CREATE operation to succeed in placing code into state.
G_{call}	700	Paid for a CALL operation.
$G_{callvalue}$	9000	Paid for a non-zero value transfer as part of the CALL operation.
$G_{callstipend}$	2300	A stipend for the called contract subtracted from $G_{callvalue}$ for a non-zero value transfer.
$G_{newaccount}$	25000	Paid for a CALL or SUICIDE operation which creates an account.
G_{exp}	10	Partial payment for an EXP operation.
$G_{expbyte}$	10	Partial payment when multiplied by $\lceil \log_{256}(exponent) \rceil$ for the EXP operation.
G_{memory}	3	Paid for every additional word when expanding memory.
$G_{txcreate}$	32000	Paid by all contract-creating transactions after the *Homestead transition*.
$G_{txdatazero}$	4	Paid for every zero byte of data or code for a transaction.
$G_{txdatanonzero}$	68	Paid for every non-zero byte of data or code for a transaction.
$G_{transaction}$	21000	Paid for every transaction.
G_{log}	375	Partial payment for a LOG operation.
$G_{logdata}$	8	Paid for each byte in a LOG operation's data.
$G_{logtopic}$	375	Paid for each topic of a LOG operation.
G_{sha3}	30	Paid for each SHA3 operation.
$G_{sha3word}$	6	Paid for each word (rounded up) for input data to a SHA3 operation.
G_{copy}	3	Partial payment for *COPY operations, multiplied by words copied, rounded up.
$G_{blockhash}$	20	Payment for BLOCKHASH operation.

Figure A.1: Table of the gas required to perform a given task in an Ethereum blockchain.

Appendix B

Enclosed Files

- **Contracts:** Contains all Solidity Contracts created for this project
- **ForeignNodeAccess:** Contains all web3.php functionality and composer for Foreign node functionality
- **JavaScripts:** Contains all JavaScripts used for the testbed
- **Simulation:** Contains all simulation scripts for this project and an emulation evaluation tool
- **root folder:**
  - **docker-compose.yml:** Docker compose file. Final report does not use this file, since emulation failed
  - **genesisTesting.json:** Genesis used for all testing in this project
  - **truffle.js:** The truffle connection script