Bachelor Degree Project

Emergency Communication for LoRaMesh using Blockchain and Distributed technologies

Author: Joakim Bjurehov
Supervisor: Daniel Toll

Semester: HT 2022
Subject: Computer Science

Abstract

In today's society we need an emergency communication system to facilitate communication for when disaster strikes. Where the previous attempts only focused on the network communication and were missing a messaging capability between users. This thesis focused on using blockchain technologies and distributed technologies to validate if a messaging application could be built for a LoRaMesh network by using these technologies. This thesis uses the Design science method to create a design and a proof-of-concept messaging application based on the LoRaMesh protocols network constraints. The first step was to gather knowledge about Blockchain technologies and distributed technologies since these were found to be able to complement weaknesses of IoT protocols and their devices. This knowledge was then used to create a design which could then be used to create the proof-of-concept application and a controlled experiment used to validate the solution. The controlled experiment was executed in two different iterations for a total of 430 test runs. From this controlled experiment quantitative data could be collected and then measured using the statistical analysis method linear regression analysis. The linear regression analysis was used to produce statistical data to validate the design and proof-of-concept application by controlling it against a standard p-value. The results and conclusions of this thesis contributed to new knowledge by showing how Blockchain technologies and distributed technologies can be used to complement each other. To gain a decentralized message application which can be used in an Emergency communication network based on the LoRaMesh protocols network constraints.

Keywords: *Proof-of-concept, Emergency communication, Messaging application, Blockchain technologies, Distributed technologies.*

Preface

I would like to thank my supervisor Daniel Toll from Linaeus University, who gave me valuable constructive feedback and inspiration which helped me improve this thesis. I would also like to thank my father Andrej Bjurehov and my college Joachim Mångren for helping me with this thesis. They both helped proof read the thesis several times and gave me valuable feedback that helped me improve it.

Contents

1 Introduction

In today's society there exists emergency communication networks for the police, hospitals and the military, for example the communication network Rakel in Sweden [1]. But the civilians also need ways to communicate when disaster strikes. There are multiple implementations of emergency networks for earthquakes and other types of disasters as shown by multiple research papers [2][3][4]. These communication networks are often based on LoRa, a long range network protocol which is used in Internet of Things (IoT) applications. In one of these emergency communication systems the developers built a LoRaMesh network which can be used to communicate in the Alps when incidents occur and in this thesis the authors state that this functionality could be extended with a messaging application [3].

There are many existing solutions to build messaging application using standard protocols, for example, Hyper Text Transfer Protocol Secure (HTTPS) which is used to send and receive encrypted data, Domain Name Systems (DNS) used to translate an IP address to a name, etc[5][6]. But these protocols do not often cover all weaknesses of Internet of Things (IoT) devices. For example, protocols like DNS or Public Key Infrastructure (PKI) which is used to facilitate how to manage cryptographic scheme in HTTPS protocol both have security vulnerabilities. The central authority for these protocols can go down, get destroyed or be unreachable caused by the vulnerabilities or natural disasters or incorrect implementations of these protocols [5][6]. These issues can cause a disruption in communication for end users and require that they rely on these central authorities to solve the problems and mitigate these threats. There exists updates or mitigations for security vulnerabilities when a device is not embedded. For IoT devices these updates need to be specially made for that specific device which seldom happens. While the mitigations are often too demanding on the IoT hardware. But for natural disaster there are no solutions when infrastructure gets destroyed by for example, an earthquake which then removes all normal means of communications [4]. But blockchain and distributed protocols combined with IoT can help creating a decentralized solution which wont be bound to a single location and help mitigating these threats in an emergency communication network [5]. De Klerk et al. suggested that Blockchain and distributed protocols can be used to complement the weaknesses of IoT devices through his theoretical study.

This thesis aims to implement an IoT compatible messaging application based on distributed protocols that can run effectively without centralization for a 15 HEC Bachelor thesis in computer science. With registration/identity management capabilities to find users based on blockchain technology that uses the proof of authority algorithms to limit heavy use of network resources in the context of IoT networks [5]. By building a proof-of-concept application using Design Science methods that can lookup identities, authorize people to join the network and facilitate messages exchange between users.

The main target of this thesis is to provide a foundation by exploring if blockchain and distributed technologies can help facilitate an Emergency communication network for an IoT network based on LoraMesh network constraints for future Computer science students or developers who wish to build on this solution. It could be used to extend a more complete implementation of an IoT network and as a resource when users need to communicate with only limited resources available.

1.1 Background

In a disaster, electricity, phone lines or Internet won't be available. In Sweden the authorities use the Rakel network to communicate and in 2019 the Swedish government started to build a way for civilians to be able to call in to this network in an event of a disaster through local firefighter stations [1][7]. This network is limited and only allows civilians to communicate with emergency personnel.

Though there are networks that could be used by civilians, for example amateur radio, the bands used are strictly regulated which is not a viable solution for civilians [8]. In recent years a new network technology called LoRa which stands for long range has been used to build robust communication networks for Internet Of Things (IoT) products [9].

From the beginning, LoRa was only used in a LoRaWAN implementation of the protocol, where WAN stands for Wide Area Network [9]. But it has also been developed into a mesh protocol which is called LoRaMesh. A mesh protocol is a protocol where all nodes are connected and they can receive communication or route it through the network since it has no hierarchy [2].

LoRa uses license-free sub-gigahertz radio frequency bands like, for example North America 865 MHz, Europe 915 MHz or 863-870/873 MHz [9]. It enables long range transmission for up to 15 kilometers in rural areas. But it also has limitations, for example bandwidth capabilities, which are between 7.8 to 500 kHz [10]. But in Macaraeg et al. research paper they show that an off-grid decentralized network for emergencies with long range capabilities is viable to develop [2].

There are also security problems when implementing an IoT network in the form of weaknesses. For example, older versions of the DNS protocol, which is used to translate an Internet Protocol (IP) address to a name or PKI, which is used to implement protocols or solutions on how a cryptographic key should be handled or even specially made protocols to handle specific use cases [5]. De Klerk et al. argues that the weaknesses of IoT can be complemented by blockchain and distributed technology. By adding a decentralized DNS based on blockchain, which can take care of privacy and security issues in the DNS protocol in context to IoT devices and their security issues, which get magnified by adding already insecure protocols to the network.

This thesis will aim to be used as part of future development around an emergency communication system based on IoT network devices where it will try to act as a base for exchanging messages securely between users in the network.

1.2 Related work

In this section we account for existing LoRaMesh solutions used for emergency communication and Blockchain solutions used to complement IoT weaknesses [5].

Emergency communication networks have been researched much using the LoRa protocol to implement it in a variety of ways using IoT devices. In Macaraeg et al. research paper they state in their conclusion: "Communication is vital especially in times of emergencies or calamities" [2]. The study implements a LoRa based Mesh network for communication during an earthquake incident in the Philippines. They focus on implementing a modified distance vector routing protocol to enable mesh capabilities in LoRa where they could transmit text messages to other phones in an off-grid network, where this implementation will focus on building a solution to find and keep track of specific users to be able to exchange messages.

Andersen et al. implemented an asset tracking solution using PyMesh [3]. They focus on implementing a LoRa Mesh for distributed position tracking using the Global Positioning System (GPS). They suggest adding a messaging system to their solution which they found to be a viable next step which is what this implementation wants to do. But focusing totally on the message exchange, how the messaging application can be designed and build a proof-of-concept application from that design. To facilitate an exchange of messages between users, the study of De Klerk et al. is interesting since it evaluates blockchain for IoT through a theoretical study [5]. Where De Klerk et al. compare DNS and PKI protocols in the context of an IoT environment. They state here that blockchain can be used to complement weaknesses in the IoT devices but some problems need to be taken care of. For example, proof of work algorithms are too cost intensive to validate that resources are part of a blockchain. They suggest building a proof-of-concept where blockchain technologies are used to handle the weaknesses of protocols like for example, DNS. This could be done by building an alternate method which does not use DNS to find a users host in an IoT environment which this thesis aims to do with blockchain using the proof of authority algorithm [11].

In J. Alsayed Kassem et al. study they show a decentralized DNS solution, which is compatible with today's DNS [12]. It's distributed on multiple nodes and used for identity management through smart contracts on a private Ethereum network to register data on the blockchain with proof of work algorithm. The registration process gives the user the possibility to control the data through the smart contract which is stored on a separate node too takes care of heavy computation loads, which blockchain can cause. The node can also act as a control to hinder malicious registrations in the blockchain network but still makes it decentralized since the users can still control their blockchain information. This is what the author also wants to implement using the proof of authority algorithm in the context of an IoT network where resources are limited.

1.3 Problem formulation

To the author's best knowledge there exists no viable solutions for message exchange between civilians in emergency communication networks based on LoRaMesh and there are no implementations of blockchain technologies in an IoT network that try to counteract the weaknesses of IoT devices [2][3][4]. For example, low security standards due to old protocols or lack of protection which exist due to the low resource capabilities [6]. This thesis will try to show that blockchain and distributed technologies can be used in an IoT network and how it can contribute to an emergency communication network. By creating a messaging application based on blockchain using the proof of authority algorithm and distributed technologies which can greatly benefit an emergency communication network. With this knowledge and above mentioned challenges in mind, this thesis will try to answer the research questions stated below:

RQ1: How can blockchain be implemented in an IoT network to build a decentralized solution which is used to find people to communicate with?

RQ2: What technologies are needed to implement a blockchain solution in an IoT network to facilitate an emergency communication network when considering the limitations on network bandwidth?

1.4 Results

This thesis presents the steps needed to create a design for a proof-of-concept message application using the Design Science method [13]. The design is created for an emergency communication network that is based on LoRaMesh network constraints [5][6]. To create the design knowledge was gathered on blockchain, distributed technologies and how these can be used to complement an Emergency communication network based on LoRaMesh. The design created in chapter 4.2 helps show how the research questions could be answered. This design is then used to create a proof-of-concept application [14]. A controlled experiment was used to gain measurable results from the proof-of-concept application showing that the answers to the research questions gives a viable solution for the chosen implementation of the message application. Then when running the controlled experiment quantitative data could be collected. The quantitative data can then be evaluated through statistical analysis method called linear regression to produce statistical data [15]. The results from the statistical analysis method were then compared to a standard p-value which could evaluate if the design and proof-of-concept application was a viable solution [16]. The results from these steps were then presented according to their generalizability, new knowledge contributed and further research that can be done with this study.

1.5 Scope/Limitation

There are several solutions which apply the LoRaMesh protocol to build an emergency communication network so this thesis will not focus on building a new LoRaMesh network application. But Andersen et al. suggest adding a messaging system to their solution which this thesis will focus on in the context of IoT networks and their limitations [3].

This will be done by applying Blockchain and distributed technologies to facilitate a secure message exchange application. The application can operate under the low bandwidth constraints of an IoT network and focus only on measuring how well the application can be used to exchange messages when applied in this context.

The application will be executed through a controlled experiment in a simulated environment with no real users where only the mean of the network propagation delay is measured to gain data to validate if the research questions can be answered [14]. The purpose of this limitation is to gain data faster and easier since there is a time frame to this project. But also to get more accurate measurements without external interference through other network traffic.

1.6 Outline

The report is organized as follows. In chapter 2 theoretical information is gathered on blockchain, distributed technologies, LoRaMesh and data to measure message application usability. Chapter 3 discusses the Design Science method, how it is used to create two artifacts, reliability and validity of this thesis and Ethical considerations. Chapter 4 first creates objectives for the design, tests and proof-of-concept application artifacts and lastly shows how it is developed. In chapter 5 the results are demonstrated through a controlled experiment from the two iterations which it took to build an application. Chapter 6 evaluates the results from the two iterations by analyzing them through linear regression. Chapter 7 discusses the results, evaluation in the context of validity, reliability and related work. Chapter 8 shows the conclusion and further work.

2 Theoretical Background

This section will go through information which was gathered when performing an overview of existing literature on blockchain technology and distributed protocols. The information gathered here is used as theoretical knowledge to build a distributed messaging system which can be used in an IoT network.

It starts by explaining the Blockchain technologies and important protocols found that are of interest to this thesis. It continues by explaining the distributed technologies. Followed by a description of the LoraMesh protocol to get an understanding of why specific test cases are chosen later to try and answer the research questions. Lastly information was gathered on message applications to try to get an overview of accepted message delay used for testing.

2.1 Blockchain

Blockchain is mainly used for cryptocurrency, with the best known being Ethereum and Bitcoin [17]. Blockchain implementations use blocks where each block is linked to the previous one through a cryptographic function called hashing and is used to organize and store data on a decentralized network. It is important for our implementation since users' information will be stored on the blockchain. It has an advantage over other implementations that are not distributed and does not require that the user trusts the party responsible for storing the data securely.

Blockchain stores the data over multiple nodes with copies of the data so the number of copies of data is equal to the amount of full nodes in the blockchain network [17]. To break this kind of system you need to take down all the blockchain nodes since the information will be distributed between all nodes on the blockchain network. For an emergency communication system this is very valuable since users will probably not be at a known location when a disaster strikes.

In a blockchain network there is not one single authority and instead no trust for a particular organization or user is needed, since all blocks in a blockchain are cryptographically linked to the previous ones [17]. Though blockchain has some drawbacks which needs to be considered or mitigated when implementing it. For example, it is sensitive to Denial-of-service (DOS) attacks where one node could overload all other nodes in the network by running too big blocks. It is also sensitive to Sybil attacks where one node controls the highest amount of resources in the network and for example, where an attacker can manipulate block transactions and move data as they wish etc. There is also the scalability issue where network resources and computer resources in the form of hardware can scale very high depending on how large each block in the blockchain is. For example, if each block is 2 bytes long, that consists of a header with length 1 byte and data of 1 byte length and if there exists 100 blocks on the chain. Then the chain would be a total of 100 bytes for each user who only needs to store the header and it would be 200 bytes for the full blockchain nodes that store the whole chain. Where headers are used to store the metadata which contains block number and a block hash. The header also contains a cryptographic nonces which is a form of random number that can't be created more than once and recorded time of when block was created [18]. The payload which contains the block data is just useful information for example, other previous blocks or transactions [17]. Where a transaction is a sender address, recipient address. amount of a transaction and a fee. For Ethereum the transaction can also contain input data which is used with smart contracts which is explained in chapter 2.1.1.

Every Blockchain has a block which is called a genesis block, this is the block that exists first in the chain and that all other blocks will be linked to [17]. The block generation will set the hardware requirements of the blockchain network since the size of the chain determines how much storage is needed.

There are mainly three algorithms, which are called proof of work, proof of stake and proof of authority [17]. These are used to validate data of the blockchain. The proof of work mines hashes and when a miner successfully mines a block, it will create a new block which can be mined. This gives the hash rate which is controlled by having a block consisting of one or more hashes. The more hashes it contains the higher the difficulty and the more miners are needed. The more blocks that are produced makes the chain longer which then sets how much computer power that is needed to successfully mine a new block. For the proof of stake algorithm people vote randomly or choose a node that gets the most votes to produce a block. The more coins someone has the higher the probability that this user will be a producer of a new coin or it will have more voting power. The voted producer then makes the blocks in a specific time. If someone votes for multiple nodes a punishment can be used to handle such misuse. Also a delegation can be used so a user can let a delegate stake vote for another user. This delegate can then share his earnings and this can be done automatically. The proof of authority algorithm is a consensus algorithm where at least 50% of all nodes need to vote if a block is valid or not. This is based on the fact that you pay with your identity and is mostly used for private blockchain networks like the one this thesis is implementing.

There are multiple types of nodes in a blockchain network [17]. Mining nodes produce blocks and have high computing resources. Full nodes are used to store the entire copy of the blockchain and they validate the blocks that they receive and send these validated blocks out in the blockchain network. Light nodes cannot store the whole blockchain but they can store the headers of transactions, blocks and need full nodes to validate block creations. The Light nodes can only be used to connect to other nodes, receive new blocks and send transactions and validate blocks or transactions by checking it against their headers.

The peer discovery in a blockchain network needs to either be manually provided, hardcoded or use DNS lookup methods to be able to find a peer in the network [17]. When sending a transaction in a blockchain network a protocol called Gossip is used. The protocol tries to spread a message to all people until everyone has the message. The node starts by sending a transaction and if another node has it, it will not download it. If the node does not yet have the transaction it will download it and propagate it to its neighbors by using the gossip protocol. This is done until all nodes have the information of a specific transaction. The gossip protocol is also used in this application but for propagating messages to other users which are not part of the blockchain.

The same kind of propagation is used with blocks. When a mining pool finds a hash it will propagate to its neighbors [17]. This is almost the same for a system using Proof-of-stake algorithm, but when it is time to create a block it collects all transactions and puts them into this block. This block is then propagated to all nodes in the network. For there to be only one of each block, the network needs to achieve consensus. This is not a problem if all the information propagated is different. But if two miners try to propagate the same block at the same time a winner is determined by the chain length. For a time two different forks will exist until one is longer than the other and the shorter chain will then be removed.

2.1.1 Ethereum

Ethereum is an open source blockchain implementation and is an alternative blockchain protocol implementation to Bitcoin which can be used for building decentralized applications [19].

Developers of Ethereum believed that it would be a useful protocol for applications that need rapid development time, security and ability to interact with different applications, etc [19]. First implementation of Ethereum was implemented with the proof of work algorithm for the main blockchain and proof of authority for test nets and private blockchains, but since 2021 it has changed its validation algorithm to proof of stake instead for their public blockchain.

Ethereum uses accounts with a 20-byte address, which is used to identify a user [19]. The accounts can include nonces, account balance, contract code (smart contracts) and accounts storage. Accounts can send transactions to other accounts and use a private key to sign their transactions. The private key is used to validate that this particular account has access to a contract or has made a specific transaction. This is important for this work since RQ1 tries to answer how blockchain can be implemented for an IoT network to find people to communicate with. Since user's identities will be based on accounts and their information that will be stored in smart contracts on an Ethereum private blockchain using the proof of authority algorithm.

There are different types of smart contracts in the world, but in Ethereum they are just considered computer programs which can be thought of as executable objects [20]. In the context of Ethereum, smart contracts can be implemented by a specific-purpose high level programming languages source code used to build special implementations of Ethereum. There are many different implementations but the one this thesis will use is Solidity since it can be compiled for many programming languages and is most widely used when developing smart contracts for Ethereum. Solidity could, for example, be used to build a decentralized application (DAPP) which is basically an application that utilizes a smart contract for a specific function. For example, Ethereum smart contracts through Solidity were used in an implementation of Namecoin which is a blockchain implementation of the DNS protocol based on Blockchain where each identity is stored in a block on the chain through the smart contract [19].

DAPPs consist of layers for the user interface which is based on a library called web3 that is used to interact with the blockchain and a layer for high level languages to develop the application [21]. Followed by a layer for storing metadata and handling lower layers of the stack. Lastly two layers handle functionality for distribution, interaction, consensus and the building blocks for networking and computations, for example, Ethereum virtual machines (EVM) [22]. The EVM can be thought of as a high level machine that is a decentralized computer which has all executable objects in a storage that is permanent and globally accessible to the blockchain network.

To build an Ethereum Dapp there are certain tools which are needed [20]. These are Solidity for building smart contracts from high level programming languages, an EVM solution like Geth which is the official implementation of an Ethereum node to execute these on a full node to keep track of transactions and computed blocks. The storage requirements and the scaling of a blockchain can handle storing and keeping track of files or other media. The EVM solution implemented by this thesis is called GoQuorum and was chosen since it removes the need for the GAS cost which is normally implemented by smart contract to keep track of transactions and avoid DOS or sybil attacks on the Blockchain [17][23]. This thesis application is on a private network where it wont have Internet access so it doesn't need to defend against these kinds of attacks.

2.2 Distributed technologies

Inter Planetary File System (IPFS) is a distributed file system which is used to store or access data, applications, websites or files [24]. IPFS is used as an example here to illustrate how content gets published in a peer-to-peer network that uses Libp2p library which is used extensively in this thesis to facilitate the message exchange of the application and is explained further down in this section.

IPFS searches for content instead of a location or an address [24]. For example, if a user queries www.youtube.com/video/funnycats in the browser, then it will search for the location of Youtube and then the content will be presented by the server. In IPFS a file instead would be searched for by its content on all nodes for video/funnycats in the distributed file system and these are called multi-addresses and can be published with the UDP protocol, TCP protocol, etc. If the content is found it can then verify it by a specific hash that is stored with the content to validate it. The pros of using this technology to find content is that every node in the peer-to-peer network can have it. It would be very redundant since it wont depend on a single location having resources that are searched for. This works by using peer-to-peer protocols to distribute its content on a network [25]. The information can be accessed by all peers in the network and the content can be distributed or read. The content on the network is also versioned and if an update resource is created it would get a new version and would get a new unique address in the file system.

Its content is linked via direct acyclic graphs (DAGs) and the content can be discovered through a distributed hash table (DHT)[26]. The DHT functionality is very important for this work since it is the implementation which is used to route messages and discover peers in the proof-of-concept application. The DAGs is a data structure which uses a Merkle tree which is a tree system of hashes, where if a hash at the bottom of the tree gets change every parent hash changes. In the DAGs each node has a unique identifier which is represented as a hash in a Merkle tree. These hashes are content identifiers (CIDs) and in DAGs everything is presented by a CID. The CID can then be found by peers if they use the DHTs. These are databases which store the hashes that are keys to values that a user can search for.

It uses Libp2p to handle peer connections and is used to search for content [25]. When a peer knows where the content is it can exchange this content from multiple nodes with a module called Bitswap which can directly connect to peers and download the content.

The data can then be verified by checking that the CID corresponds to the downloaded content. Every peer also has their own CID which is their unique identity and is used to represent the nodes in the network.

CID's are also used in Ethereum blockchain when representing a user and also by the Libp2p application to represent a host. The content of the network is stored on the data nodes which can be a DAG node and it is here that files can get uploaded. Then there are IPFS nodes which are essentially peers, daemon or instances in the network which are used to access data in the IPFS network.

Lastly there is Libp2p peer which is the library that can be used to build applications for a peer-to-peer network by accessing data or publishing content [27]. It has the ability to route or multiplex data or publish and subscribe. The publish and subscribe method of the library is what will be used to build a decentralized chat application for this thesis, together with the possibility of publishing addresses through CID's and using DHT for discovering new nodes [26].

2.3 LoRaMesh

The protocol LoRa which stands for long range, has been used to build robust communication networks with Internet Of Things (IoT) products [9]. It is important for this thesis since the resource constraints are used for testing if the application can be implemented with this protocol.

It uses a spread-spectrum modulation technique which is created from the chirp spread spectrum (CSS) technology [9]. CSS uses a wideband linear frequency modulated chirp pulses to encode the information that is sent. It uses the entire allocated bandwidth to broadcast signals, which makes it robust against channel noise. It also has 12 different settings to set its CSS depending on the range, which is needed when sending messages.

LoRa uses license-free sub-gigahertz radio frequency bands like North America 865 MHz, Europe 915 MHz or 863-870/873 MHz [9]. It is possible for it to get a range of up to 15 kilometers. It has been implemented to handle mesh networks and those networks can be fully decentralized with border routers, which can connect to other types of connections, for example Ethernet, mobile networks, etc.

There are different IoT network devices which have implemented the LoraMesh network protocol. But the one that is interesting for this thesis is PyCom's PyMesh which can be run on their Lopy4 hardware [10]. This hardware has some limitations though, for example, its bandwidth can be between 7.8 to 500 kHz.

Though bandwidth expressed in kHz is not useful in this thesis since it needs to be measured and a bitrate needs to be established. This can be calculated by taking the noise ratio and bandwidth into consideration. In LoRaWAN Mesh Networks: A Review and Classification of Multihop Communication the author shows how bandwidth can be translated to bitrate. Here we get a span between 4913.7 bit/s and 146.1 bit/s which are the highest and lowest values [28]. This is calculated from a bandwidth of 125 kHz/s with a signal noise ratio based on LoRamesh CSS technology. This will affect the controlled experiment in that it will be one of the constraints applied when validating if the application can be implemented on a LoRaMesh network.

2.4 Message application

To be able to evaluate the research questions on this thesis an acceptable performance metric for message applications was needed. So data was gathered on what users would deem as an acceptable network delay in order to get a clear measurement of what is acceptable for a chat application.

A message application is a non real time service and is used to stream audio or messages between users [29]. These applications have acceptable network delays of less than 5 seconds for an end user according to a study made on Delay Limits on real time services. This is also one of the dependent variables used to measure if the application can be implemented in a LoRaMesh network and still have usability.

3 Method

In this chapter a method is described to answer the research questions from introduction chapter 1.3. By using the methodology Design science and a controlled experiment which will be explained in this chapter [5][14].

RQ1: How can blockchain be implemented in an IoT network to build a decentralized solution which is used to find people to communicate with?

RQ2: What technologies are needed to implement a blockchain solution in an IoT network to facilitate an emergency communication network when considering the limitations on network bandwidth?

The research methodology design science is based on six steps and is used to create something new that doesn't yet exist. This thesis uses the method to create a design for an application and tests for the application. Then the design gets implemented by creating a proof-of-concept application.

Next a controlled experiment is used to test the application through a controlled environment and measure the data that it generates [14]. This will measure until the amount of users using the application reaches the maximum acceptable network delays of 5 seconds, which is refereed to in chapter 2.4. The data collected by the experiment can then be evaluated through a statistical analysis method to validate that the results are not found by chance [15].

Then if the results where not found by chance it can be used to answer the research questions by showing if the chosen blockchain implementation and technologies for the message application is a viable solution for a certain amount of users.

3.1 Design Science

This research will use design science research methodology (DSRM) based on Peffers et al.'s description [13]. Design science is used to produce an artifact which is a solution to a real relevant problem, it can be an algorithm, code, a design, proof-of-concept, etc. It suits the problem of designing a messaging platform for an IoT network. Because design science is a research paradigm that can contain many methodologies for creating something new and help to prove or disprove that it can solve the problem. The design science methodology consists of six steps shown in the flowchart below [13].
The first step by Peffers et al is problem identification and motivation. It is where the research problem is defined and justified [13]. Then break it up into smaller problems that can easily be handled to construct an artifact. It should show that the problem is well understood and motivate why the specific solution is being pursued.

The second step by Peffers et al is to define the objectives for a solution by using the problem definition, related work and knowledge about a solution [13]. Here the method tries to divide the problem into what type of objectives the solution has, for example if it is qualitative or quantitative.

Note that the following steps, Design and Development, Demonstration and Evaluation will be done iteratively [13].

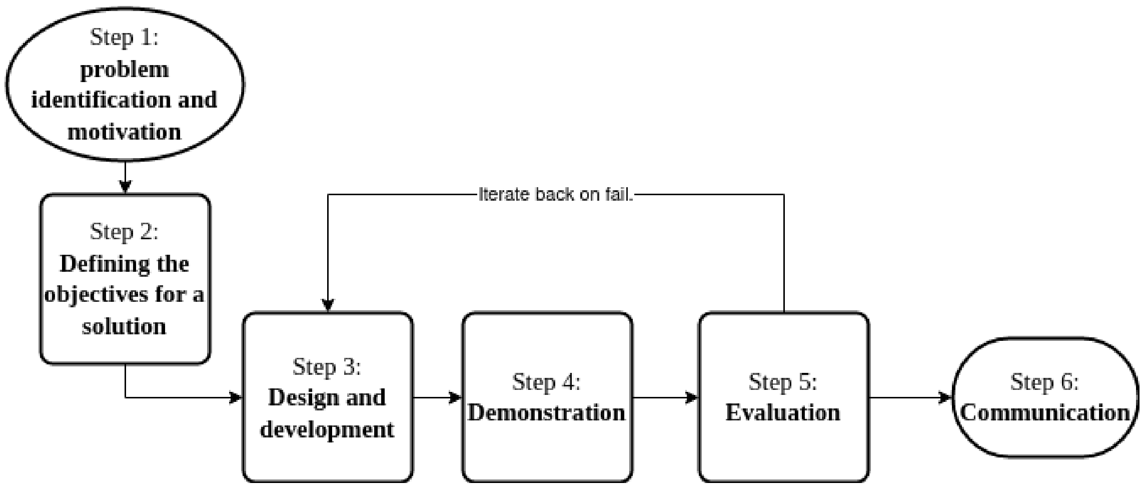

Step 1: problem identification and motivation
Step 2: Defining the objectives for a solution
Iterate back on fail.
Step 3: Design and development
Step 4: Demonstration
Step 5: Evaluation
Step 6: Communication

Figure 3.1: A flowchart over Peffers et al. implementation of Design science research methodology.

The third step by Peffers et al is design and development. This is where the artifact is designed and created based on the requirements from the objective phase[13].

The fourth step by Peffers et al is demonstration, where it can show that it solves the problems or some of the problems defined [13]. This could be demonstrated by using, for example, an experiment, simulationproof, case study, etc. Demonstration data will be gathered to gain a result which will then be evaluated in the fifth step.

The fifth step by Peffers et al is evaluation where the artifact can be observed, measured and it can be seen how well it supports the solution of the problem [13]. Here the data is gathered from running the solution in the demonstration phase. Relevant knowledge of the metrics and analysis techniques are required so that it can be measured correctly. This step decides if the artifacts need to be refined by iterating back to step three or if the results are adequate and should be reported in the next phase.

The sixth step by Peffers et al is communication, where the results of the solution to the problem is reported and the artifacts contribution is communicated to the relevant audience [13].

3.2 Research overview

In this chapter our implementation of design science is used to answer the research question by defining how two artifacts are created through the method. Artifact one is a design and artifact two is a proof-of-concept application messaging system that will run in an IoT network. This application will need to be designed and evaluated in a manner that answers the research questions of this thesis. The methodology consists of six steps described in chapter 3.1 and these steps are implemented as shown in the figure 3.2 below [13].

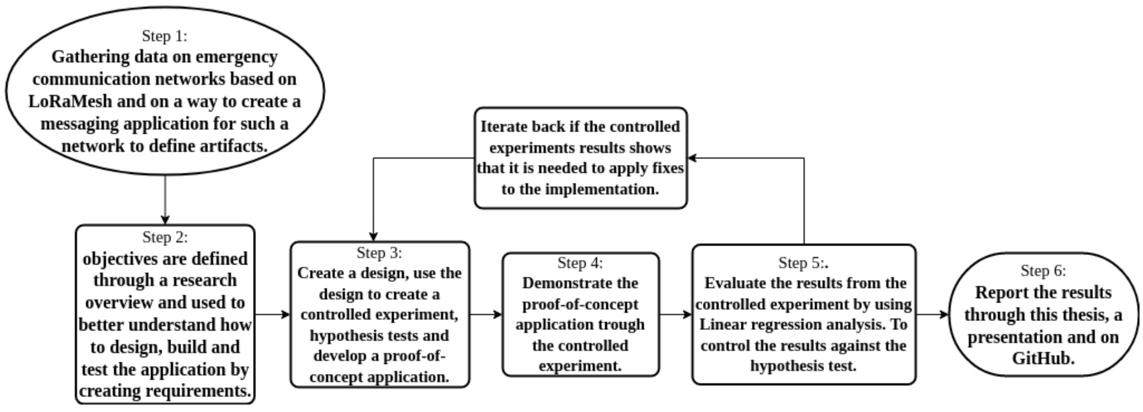

Step 1:
Gathering data on emergency communication networks based on LoRaMesh and on a way to create a messaging application for such a network to define artifacts.
Step 2:
objectives are defined through a research overview and used to better understand how to design, build and test the application by creating requirements.
Step 3:
Create a design, use the design to create a controlled experiment, hypothesis tests and develop a proof-of-concept application.
Step 4:
Demonstrate the proof-of-concept application trough the controlled experiment.
Step 5:.
Evaluate the results from the controlled experiment by using Linear regression analysis. To control the results against the hypothesis test.
Step 6:
Report the results through this thesis, a presentation and on GitHub.
Iterate back if the controlled experiments results shows that it is needed to apply fixes to the implementation.

Figure 3.2: A flowchart over the implementation of Peffers et al. Design science research methodology.

In the first step of design science, problem identification and motivation is done by gathering data on emergency communication networks based on LoRaMesh and on a way to create a messaging application for such a network [13]. The data is then used to create the background, related work and problem formulation where research questions are defined according to related work as shown in chapter 1.2. With the help of the information gathered here two artifacts are identified. The first is a design of the application and from this design a second artifact in the form of a proof-of-concept application is created, which is used to answer the research questions.

In the second step of design science, objectives are defined through a research overview [13]. This is conducted in chapter 2 to gather understanding about blockchain, distributed technologies, LoRaMesh and messaging applications[5]. The information from chapter 2 is used to better understand how to design, build and test the application. This information and the research questions are used to divide the research questions into requirements in chapter 4.1.

In the third step of design science, the design and development phase are where two artifacts are created, as shown in chapter 4 [13]. First a design artifact is created based on the requirements from the objectives phase and this will be a logical design of the application. To be able to test the design and application artifacts, tests will be defined based on the requirements created from the research questions. These tests will be stated as hypotheses to be able to test them through a controlled experiment in the demonstration step [30][14]. Then from the design artifact a proof-of-concept application artifact is implemented to try and fulfill the requirements and answer the research questions.

In the fourth step of design science, called Demonstration is where the results are shown in chapter 5, a controlled experiment is used to show this, see chapter 3.3 [13][14]. A controlled experiment uses the independent variables which are variables that affect the value of the dependent variables that we want to measure. The controlled experiment will be implemented in chapter 4.3.

In the fifth step of design science, the Evaluation is used in chapter 6. The evaluation step will use hypothesis testing which is a statistical analysis technique and is described in chapter 3.4 [13][15]. The statistical tests are used to evaluate if it is needed to iterate back to step three or if we can continue to step six. The data gathered from the controlled experiment will then be used in the evaluation phase to check if the results should pass or fail through a linear regression analysis test [14][31]. If it fails then iterate back to the design and development phase to readjust the design or the application.

In the sixth step of design science, Communication is used to report the results through this thesis in chapter 7, chapter 8, a presentation and on GitHub [13].

3.3 Controlled experiment

A Controlled experiment was used to evaluate how well the proof-of-concept application can perform when applying the requirements from the Design science objectives step from chapter 4.1 [13]. Since a design created in chapter 4.2 where showing how the research questions could be answered. The controlled experiment shown below was where measurable results showed that the answers to the research questions gives a viable solution for the chosen implementation of the message application.

- To be able to measure how well the application works when implementing the blockchain solution for RQ1 to find a user.
- To be able to measure the chosen technologies of this implementation for RQ2 when using blockchain and considering the limited bandwidth of 4913 bit/s using an IoT network for RQ2, see chapter 2.3 for further information on bandwidth limitations of IoT.

The controlled experiment uses the theoretical knowledge gathered on blockchain technologies, distributed technologies, LoRaMesh and messaging applications from chapter 2 to create a test. Were it measured how a dependent variable was affected by the independent variable and tried to control all other variables which could affect the dependent variable [14]. The controlled experiment can also use knowledge gained from previous iterations to determine how the hypothesis test should be changed if needed. From this knowledge a question was asked as shown below.

$$When\ will\ the\ amount\ of\ nodes\ using\ the\ message\ application\ affect\ the$$
$$network\ propagation\ delay\ of\ a\ message\ so\ it\ gets\ higher\ than\ 4500\ ms$$
$$with\ a\ limited\ bandwidth\ of\ 4913\ bit/s?$$

From this question independent variables that can affect the value of the dependent variables were created for the hypothesis test to gain quantitative variables that was measured [14]. These independent variable Nodes and dependent variable Network propagation delay were used to state a null hypothesis and an alternative hypothesis as shown in the table 3.1 [14]. The variable Nodes measured the amount of users using the application. The variable Network propagation delay was used to measure when a message has been distributed between all Nodes in the network until the acceptable network delay of maximum 5 seconds was reached for a message, see chapter 2.4 for further information. These variables together measured how the amount of nodes interacting through the message application can affect the Network propagation delay. A null hypothesis and an alternative hypothesis as shown in the table 3.1 was created so they could be evaluated through a statistical analysis method [30][15].

Table 3.1 also shows how the hypothesis test below produced two extraneous variables [32]. These two variables are published messages and the amount of delivered messages. The two variables will affect the network propagation delay since every node will publish a certain amount of messages and these will need to be successfully delivered. Otherwise this will affect internal validity since network propagation delay can not be measured correctly. To control the two extraneous variables and validate that the network propagation delay is correctly based on the amount of messages sent for that test. The published and delivered messages will be calculated on each test run and included in the raw data so that results can be validated through descriptive statistical analysis as shown below in table 3.1 [15].

Variables	Hypothesis	How to control
Independent variable: The amount of nodes using the application.	Null hypothesis: Increasing the nodes using a chatroom over a certain point will not cause the network propagation delay to increase in the network for a message.	Check that at least ninety five percent of the published messages get delivered each test run. By controlling messages received through this calculation shown below: $Percentageof\, delivered\, messages = delivered\, messages \,/\, (published\, messages * nodes)$ The result can then be evaluated after each test run. So that ninety five percent of the delivered messages for each test run can be confirmed. Then the data can be presented through a histogram [15].
Dependent variable: Network propagation delay	Alternate hypothesis: Increasing the nodes using a chatroom over a certain point will cause the network propagation delay to increase in the network for a message.	
Extraneous variables: Published messages and Delivered messages.		

Table 3.1: The tables show the independent variable nodes, how these can affect the dependent variables network propagation delay and how these are used to state the hypothesis test. It also shows the Extraneous variables and how these are controlled.

To implement the controlled experiment a virtual environment is used based on VMware ESXI [33], see chapter 4.3 for further information on the implementation. Were the unknown variables like hardware failure can be avoided and hardware resources can be closely monitored.

The results from the controlled experiment were then evaluated by using a statistical analysis method called linear regression analysis to produce a p-value which can be compared to a standard p-value of 0.05 [31][16]. Showing that the null hypothesis can be rejected, if the obtained p-value is lower than, or equal to the chosen standard p-value.

3.4 Reliability and Validity

Internal validity is very important for this research since it would show that a cause-and-effect relationship between two variables is trustworthy and reliable [34]. Validity specifically means that a method successfully measures what is supposed to be measured. High internal validity in this case is needed when measuring how the amount of nodes in the network will affect the network delay to be able to answer the research questions.

Internal validity will try to be strengthened by using a controlled experiment which will be executed in a virtual environment as described in chapter 3.3 [14]. The virtual environment will also make the controlled experiment easier to reproduce by others, which will strengthen the reliability of the results. But by using a virtual environment instead of a real environment, external validity is weakened since it is not generalizable to a real environment, which has other variables affecting the results. For example, distance between nodes when using the LoRaMesh protocol cannot be measured in a virtual environment.

Reliability is also important to this thesis and it is used to show how exact a method can measure something [34]. The measurement can be considered reliable if the same circumstances and method can achieve the same results. The reliability will be strengthened by using a known platform for testing called Testground, as described in chapter 3.3. Reliability will also be strengthened by having the results from the test, source code of the application, environment setup and test setup documented on GitHub so that anyone who wishes to test the application can reproduce the results.

There are variables in this thesis that are not specifically investigated but can still affect the results of this thesis, these are called extraneous variables [32]. In this thesis two extraneous variables were identified that can affect the dependent variable by yielding an incorrect result for measured network propagation delay. The variables identified are the amount of published messages and the amount of delivered messages [35][36]. To strengthen the internal validity of the network propagation delay results, these extraneous variables will be measured and calculated[32]. The calculation needs to show that at least ninety five percent of the messages get delivered each test run to include the results from a test run, see chapter 4.3.

The tests will be limited to only testing how well the application can perform under IoT network constraints that exist for the LoRaMesh protocol [28]. This will strengthen internal validity on the results but weaken the external validity since it won't be generalizable [34]. External validity of the results is also weakened since each sample group will have some amount of randomness that will cause a sampling bias [37]. This is due to the gossip protocol being random when it chooses through which path a message gets delivered in the network by always having a peering degree of 6 peers, which are directly connected [38][36]. This may affect the measurements by showing an overly positive result or a negative result. For example, the nodes can randomly take the fastest path or the worst path to deliver a message. To make the results more generalizable, oversampling can be applied to try and gain all the data points, since it mimics the protocols real behavior [37][14]. This can then strengthen the external validity of the measurements associated with this controlled experiment. Oversampling will be applied by running each test group at least four times so the validity of the conclusion and reliability in the results can be strengthened.

The statistical analysis method used here strengthens the internal validity of the results since they can help reject the null hypothesis [15][30]. The tests are implemented in the form of first representing data in a graph through descriptive statistics to check that the results are reliable and work with linear regression [31]. This is done by checking that the values are normally distributed across the graph by checking that they follow a roughly shaped bell curve. They should also form a linear pattern which is shown by having the results presented through a scatter plot. Also checking that the results of the tests are independent observations. This is ensured by running all test runs in an environment that has been reset to default values before the next run. The results have no hidden relationship between each other since only one dependent and independent variable. Then checking that the model meets the assumption of homoscedasticity of the results by having them calculate the residual unexplained variance, which should not change significantly over the range of predictions in the model.

The inferential statistics are used to measure how well the data performed through a linear regression test to calculate the p-value and checking that the results were not obtained by chance [16][34][31]. This can be used to strengthen the reliability and validity of the results by showing that the data fits the model well and has measured a cause-and-effect relationship between the independent and dependent variables.

3.5 Ethical considerations

The application is a proof-of-concept and there exists many implementations of these types of protocols. But not where Ethereum is used for an identity and authorization service together with a messaging app in an IoT network.

Ethical considerations can be made while examining the context of how the application operates when it finds and communicates with other peers. It sets up a network that is easy to use and sends messages where everyone can be used as a router [36]. This makes it hard to trace the communication since there is no centralization, this would be perfect for criminal organizations who wish for no centralized storage of their communication and this type of usage has been seen in other open source messaging applications, for example Encrochat [39].

This can be considered an issue but it depends on how people implement the application. The benefits of having a distributed messaging application in an IoT network outweigh the misuse scenario since the application can help deploy a communication network for people who have no electricity or who have no working network connection.

4 Research project – Implementation

This chapter is used to design, develop and implement a solution which can be used to answer the research questions from chapter 1.3. Chapter 4.1 is based on the second step by Peffers et al to define the objectives for a solution and will be used to define requirements for the artifacts [13]. In chapter 4.2 a design is created and in chapter 4.4 a proof-of-concept application is created based on the third step Design and Development by Peffers et al. The requirements will also be used to implement a controlled experiment for the proof-of-concept application in chapter 4.3.

4.1 Defining the objectives

In this chapter requirements are defined for the design and proof-of-concept application. The requirements for the artifacts are based on information gathered from the problem formulation and research questions in chapter 1.3. It also uses the theoretical knowledge acquired in chapter 2 to define the requirements shown below.

R1: Proof of authority algorithm for Blockchain needs to be implemented instead of proof of work so that it can be implemented in an IoT network without creating too high network load. This requirement is based on De Klerk et al.'s study where they suggest implementing another protocol than proof of work to counteract high network and computer loads[5]. This requirement will try to answer RQ2 from chapter 1.3, since it tries to solve the blockchains validation algorithm choice.

R2: Users need to be able to send messages to each other without having a network propagation delay higher than 5000 ms. This requirement is based on information from chapter 2.4 on optimal network delay for a messaging application which is less than 5000 ms [29]. This requirement is meant to test how well the application can be implemented based on the research question RQ1 and RQ2.

R3: Users need to be able to send a message to other users in a disaster without depending on a centralized location, for example, a server which holds all resources necessary to send a message. This is based on de Klerk et al. suggesting that Blockchain and distributed protocols can be used to complement the weaknesses of IoT devices [5][6]. The requirement stated here tries to answer RQ1 by taking into account that the application needs to be decentralized.

4.2 Design

This chapter consists of first creating the design for the application by describing a hypothetical scenario based on what the design is built for and what it will be used to build. This design will be presented through text to describe its functionality goals and visualization to show how different services are used in the application. Followed by text examples that will state how the application is aimed to be created and with what functionality.

The design will use LoRaMesh as its network protocol for the IoT devices [28]. It can include one or more different LoRaMesh networks and then be divided by using a border router to interconnect them to avoid over consuming the bandwidth. Each LoRaMesh network will consist of one or more IoT devices used to send and receive data in the network. Clients will access these IoT devices through a wireless protocol and the IoT devices will be interconnected through LoRaMesh protocol. The IoT devices that will act as a border router will have a Raspberry Pi connected to it. The Raspberry Pi will run one of many Ethereum full nodes as a server component in the LoRaMesh networks [23]. The Ethereum full node is based on GoQuorum, see B Appendix 1 to learn more about how to set up the GoQuorum as an Ethereum full node [23]. The Ethereum full node will use smart contracts to validate clients who wish to join the peer-to-peer network. Clients will need to be implemented as light blockchain nodes and use a peer-to-peer protocol to exchange messages to achieve decentralization [19]. The full node will be used to store the blockchain and it can also distribute it to other full nodes in the network. Light nodes need to be implemented so they can send transactions and have their own wallet to store their private key which is used to identify a valid client in the network.

The applications functionalities are described below and represent the design. It is based on the requirements from chapter 4.1 and the theoretical information gathered from chapter 2.

- The Registration service so a user can set up an account through registration, be granted access to the messaging applications chat, gain access to other users public data and the routing capabilities.
- The Verification services are used to verify who is allowed to change a user's data and to verify or claim an identity of a user.
- The Messaging services is used to receive or send a message.
- The Blockchain Getter service is used to interact with the blockchain's smart contract to get the information stored on the Ethereum full nodes. For example, get a username mapped to a user's node address, etc.
- The Client Ethereum service is used to get valid public information from the blockchains smart contract through the Getter service.
- The Routing service is used to find other users and route sent messages to each other. In the previous design from Iteration one this was split between a bootnode on a seperate server and the clients routing service. Were the client routing service queried the bootnode server for a list of bootnodes to get the routing service working. But to avoid a bottleneck which increased message delay this was integrated into the client by using the Ethereum node as a distribution point for the bootnode list.
- User interface (UI) Service exists so that users can interact with the application.

The services above will try to implement requirements R1, R2 and R3. These services are visualized as a diagram to illustrate the design as shown below in figure 4.3.

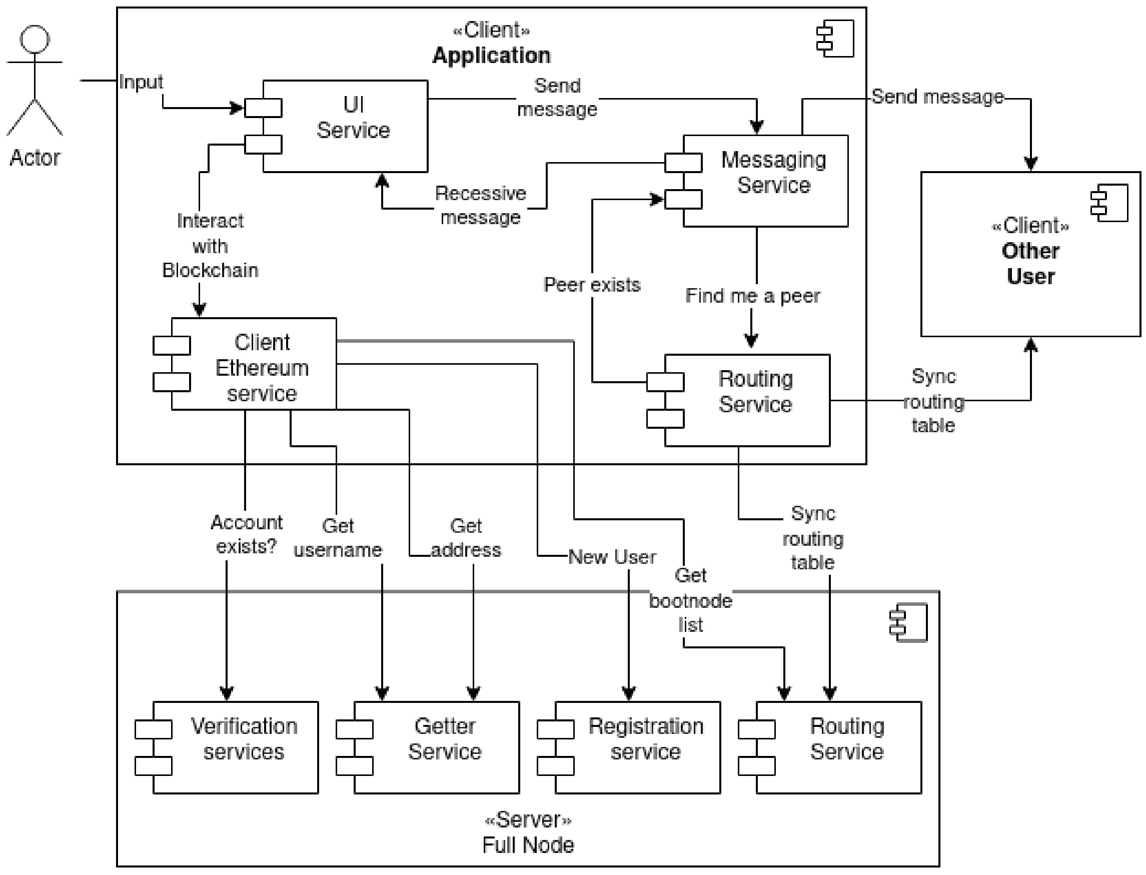


Figure 4.3: The design shows the client application, its different modules and how they interact with the server to register or get information through the Getter service, etc.

The design in figure 4.3 uses a basic Client/Server architecture to represent the design choices, but it is implemented as a peer-to-peer application. Arrows in the design from figure 4.3 shows who initiates the communication by which direction the arrowhead points. For example, Client Ethereum service initiates the Getter service by calling the Get username.

The design in figure 4.3 uses an actor to show the user of the application that can interact through a user interface on their own machine. The user interface is called the UI service that interacts through the TCP protocol with the Ethereum smart contracts through its client interface. Actors using the Message service have a send and receive function using the TCP protocol. The routing service using TCP is implemented directly at the client side and uses the Ethereum service to distribute bootstrap node lists that consist of other clients multi-addresses [26].

The Ethereum full node smart contract will be implemented through a private Ethereum network as shown in figure 4.3 [20]. The decentralized part of the Ethereum network is handled automatically if another Ethereum full node is added to the blockchain network with the same smart contract capabilities. Though since this is a small proof-of-concept implementation this won't be done with more than one server acting as a full node. The smart contracts functionality will be implemented by using the Solidity programming language as described below.

- Registration service takes a username, its unique content identifier in the Libp2p network and an address of the registered node [36]. It then checks if the user is already registered and if not then the service pushes the new user's credentials to a list on the Ethereum smart contract [20]. It then calls another function to add the user as a boot node which uses the address of the node as input.

- Getter service for username takes a content identifier as input. Then it uses a hash tree which has the content identifier as a key to search for a corresponding value which would be the username [19]. If it is a match the username is returned.

- Getter service for users node address takes a username as input. Then it uses a hash tree which has the username as a key to search for a corresponding value which would be the node address [19]. If it is a match the node address is returned.

- Getter service for the bootstrap node address list just returns an array of addresses that are registered as Distributed hash table (DHT) bootnodes [26].

- Routing service function is used to add a node as a bootnode to the network. It first sets a timestamp of the current time and it checks if it has passed 60 seconds since it was last modified [20]. Then if 60 seconds have passed the DHT bootstrap node list is set to zero and while also resetting the counter that keeps track of time [26]. Then it pushes this node as a boot node to the array of DHT bootnodes. If it has not passed 60 seconds it checks if the list is longer than three nodes and if true removes the oldest one from the list. So that the DHT bootnodes are rotated and the load of handling new nodes joining the network is spread between nodes in the network. Then it adds the new address to its list of bootnodes by pushing it in on the front of the array.

- Verification service takes no input and instead uses the blockchain ID that is sent with each transaction [19]. It checks if the sender exists by checking if they recently have registered and returns true if they are found. Otherwise it will return false if no user has been found since they are not registered. This service can be used to check if a client is allowed access to the messaging service.

The following points will describe how the client side of the application interacts with the environment through its services as described in figure 4.3.

- Routing Service will be implemented by using a Distributed hash table (DHT) algorithm which is described in the Theoretical chapter 2.2 [26]. The DHT algorithm will be applied since it uses content identifiers to search for other nodes and it only needs a boot node the first time it joins the network or if the routing table is empty which can be satisfied by having the Ethereum smart contract store a list of active bootnodes [26][20]. When a client joins the Ethereum network or if it uses it after registration it will register itself as a bootnode and use the current list to fill its routing table with active peers in the network.

- Client Ethereum service starts by first setting up an Ethereum client by implementing a wallet since all nodes need one to store their private key which acts as their identity [19]. It needs to be able to check if a wallet for the client exists, then use it if it does exist. Otherwise a wallet needs to be created and stored on the disk through a password.

Ethereum clients need to be able to send data to the blockchain and retrieve data. First it checks if a user is registered through the Verification service and is able to reject users who try to register twice with the same account by checking if they already exist on the blockchain [19]. It then needs to be able to send a registration request to the smart contract at the full node by sending what Ethereum calls transactions.

Next the client needs to be able to register as a bootstrap node if it joins a network without registering for the first time [26]. This is used if a valid account already exists and it also enables the client to get a list of bootnodes so that routing can be set up.

The client also has to be able to display who is active in the chat by using a list of usernames that are displayed in the user interface. This information is obtained by querying for a list of peers with their content identifiers translated to usernames.

- The messaging service will have to send and receive messages in the network when a user joins a chat based on the gossip protocol [38]. It will use a publish and subscribe method to handle this and this is implemented through the gossip protocol which will use TCP as a transport for communication. It will have to set up a discovery service which is connected to the routing service that uses DHT [26]. Clients can interact with the routing service and get peers' content identifiers so it can send all users a specific message which is part of a chat. It will also redistribute other clients messages depending if the client is close to another client and help with message distribution in the network since it will use peer-to-peer features [38].

- The user interface (UI) service will be where everything is taken together and presented to the users. First the Ui needs to handle user registration. The registration will be implemented by having a user input data in the terminal. This will create a wallet through the Client Ethereum service [19]. When a user has created an identity by using a self generated private key stored in a wallet. The user can join a chat where their name will be shown by having the application query the Getter service for a username for the specific user identity generated by a private key.

  Next when a user joins a chat the UI service needs to take user input from the terminal. While at the same time being able to send a message to all other users who are part of a chat and at the same time receive messages from users in the chat. It also needs to query for content identifiers the first time it is executed or if the routing table is empty [19]. This is used to keep a peer table updated of which people joined the chat and then query the Getter service for the username of each content identifier for every new participant in the chat. The peer table is implemented through a hashmap so that already found peers get stored in the hash map and only new peers get queried through the Getter service so that network loads are lower.

4.3 Controlled experiment implementation

In this chapter the experiment is implemented through the hypothesis test from chapter 3.3 and the requirements from chapter 4.1 [30]. This experiment gets implemented through a virtual environment, see A Appendix 1 for further information on the virtual environments hardware specs and setup.

The experiments sample size is based on the amount of nodes which in turn is based on the Gossip protocol's default configuration [27][38]. The default configuration tries to have at least six directly connected peers for each topic with a max of twelve directly connected peers before they start to remove peers from the routing table. So the minimal sample size of nodes for the first iteration was set to 15 nodes to avoid having all nodes directly connected. This was then measured for up to 105 nodes that were split into 7 different node groups that are incremented with 15 nodes between each group. A new starting point was found from iteration one, see 5 for further information. With the new starting point set to 10-105 nodes, sample size will be split into 13 different test groups. Where each step will be between 15 or 5 nodes increment for each test group depending on how it affects the network propagation delay of a message. This will then be measured until a network propagation delay of a single message can be observed to be 4500-5000 ms when testing the proof-of-concept application.

To test the hypothesis a experiment will be set up in a virtual environment with a network layout as shown by the figure 4.4.

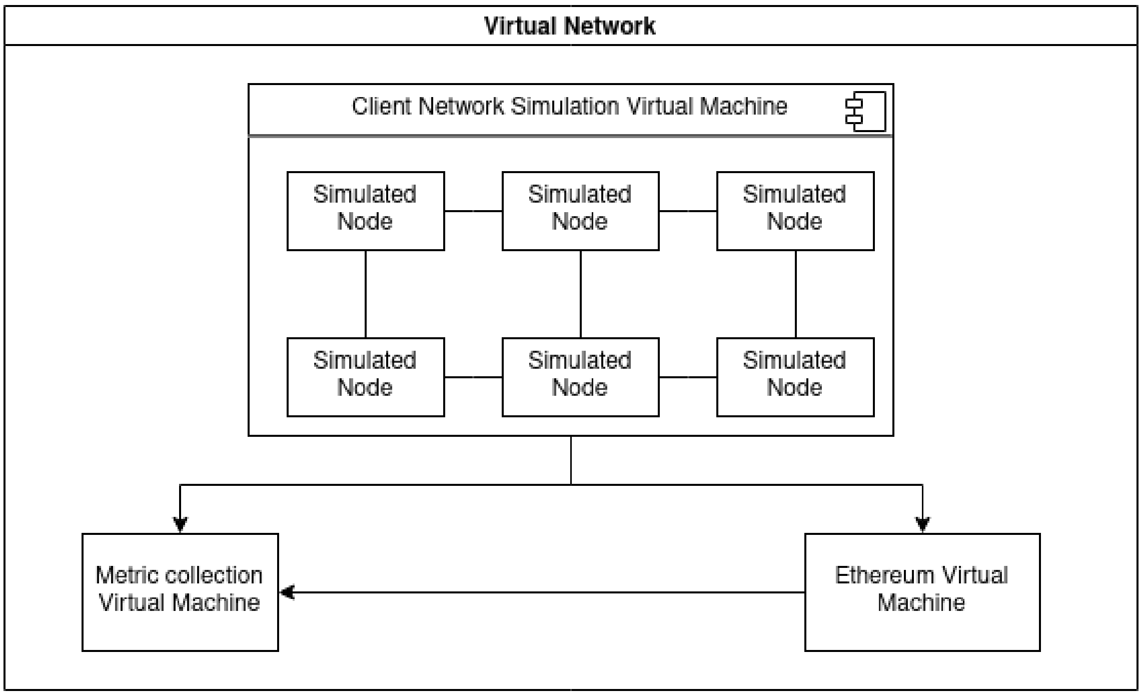


Figure 4.4: The diagram shows three virtual machines in a virtual network where each has a different purpose according to the Controlled experiment in chapter 3.3.

Then according to figure 4.4 the Ethereum virtual machine will be implemented by using GoQuorum where it will be running the Smart Contract that handles registering and mapping users to their public information [23][20]. Also to simulate a real environment as closely as possible the Ethereum virtual machine will be virtualized on a separate host [19]. This is set up this way to handle the high computer load of blockchain applications as recommended in Alsayed Kassem et al. 's research [12].

Next according to figure 4.4 the Client network simulation virtual machine will be implemented through Testground. The Client Network is a simulated network that starts Docker containers for each client through Testground and is only limited by the hardware resources of the host [40]. Testground has been used by Protocol labs to evaluate their newest version of the gossip protocol and is implemented to simulate the messaging application of this thesis [35]. Where every client is a DHT boot node that uses gossip protocol to send messages between each other and interact with the Ethereum virtual machine through their wallets holding their unique ID's [26][38][19]. Testground will also handle setting the amount of bandwidth, the length of the test run and how often messages will be sent. The simulated nodes also send information to the Metrics virtual machine as shown in figure 4.4 [40].

The metrics virtual machine in figure 4.4 will be using the go-libp2p-pubsub-tracer to collect the metrics on each message sent or received [41][42]. This collection technique was suggested when the gossip protocol was evaluated with Genesis where they evaluated a baseline for the Libp2p publish/subscribe implementation [41].

When all the results from the controlled experiment are collected a mean of all messages delivered can be calculated. Used to represent that the variable node and variable network propagation delay has a cause-and-effect relationship. This is done by using the formula:

$$(sample1\ +\ sample)\ /\ samples\ count$$

For example, if the propagation delay is 200 ms for 1 message and 300 for 1. Then if you calculate the sample (200 + 300) / 2 which would then give an average network propagation delay of 250 ms spread on all high value delays found.

4.4 Development of proof-of-concept application

This chapter will show how the proof-of-concept application in this thesis chooses to implement the smart contract and messaging application. It will be based on the design from chapter 4.2 and knowledge gained from the theoretical chapter 2 to implement this application. The application gets implemented through the Golang programming language used to develop the message application and Solidity programming language used to develop the smart contracts for the Ethereum blockchain [43][36].

The full source code for this chapter is presented through Github. Together with how to install Golang, Ethereum GoQuorum node, the test environment in the form of Testground, compile and load the smart contracts to the Ethereum chain through a Golang application, as seen in B Appendix 1 [23][40].

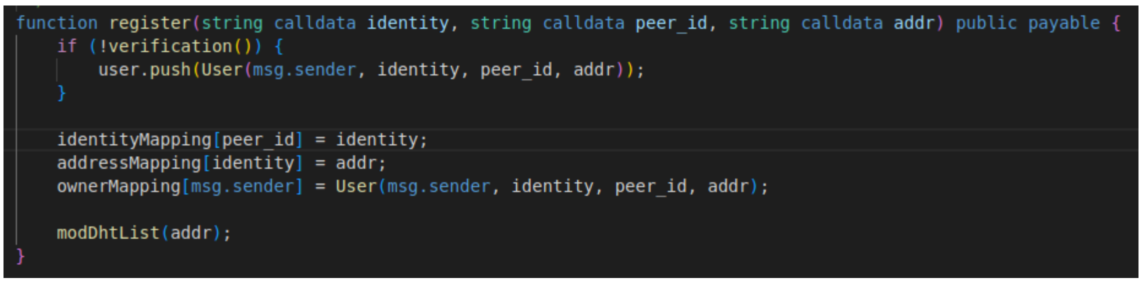
4.4.1 Smart contract

First the smart contracts are developed in Solidity through the Ethereum online integrated development environment called Remix and created for the Ethereum blockchain through the guides on Ethereum development in Golang [43]. Remix was chosen since the smart contracts can be compiled and deployed directly on test chains to observe the results when interacting with the smart contracts [44]. The Golang programming language for this application was chosen because when compiling the smart contract, optimized

Golang source code can be created for the specific smart contract [43]. The compiled smart contracts also create interfaces in Golang that can be used when interacting with the blockchain.

The State.sol package is used to implement the smart contract through a structure to hold the values of an Ethereum wallet address [43][19]. It also contains the identity of a user, a content identifier from the peer-to-peer network and their multi-address from the network [43]. The conclusion from iteration one also adds a global variable to store DHT bootstrap node addresses [26]. Then three hashmaps are created to map a content identifier to a name, the name to a specific network address and an owner to their specific User structure [20].

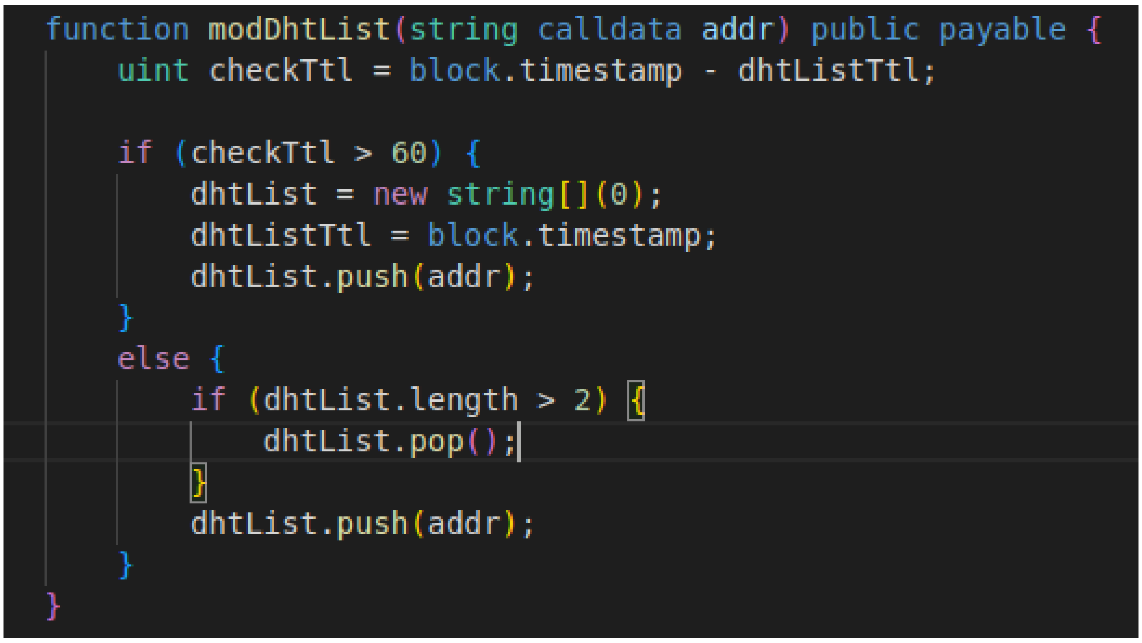
The registration service is created as shown in figure 4.5 below. The register service is publicly payable, which means that when a user registers they need to pay. The payment is done by using gas in an Ethereum network and shows how much computer power is used for that specific transaction [19]. The service takes user input to be able to map a user to their peer id and Ethereum account identity. Then the function controls if this user is registered by calling the verification function. If verification shows that a user is already registered it will stop. Otherwise it adds a user to all the structures and hashmaps needed to complete the registration process.



```
function register(string calldata identity, string calldata peer_id, string calldata addr) public payable {
    if (!verification()) {
        user.push(User(msg.sender, identity, peer_id, addr));
    }

    identityMapping[peer_id] = identity;
    addressMapping[identity] = addr;
    ownerMapping[msg.sender] = User(msg.sender, identity, peer_id, addr);

    modDhtList(addr);
}
```

Figure 4.5: This shows the registration service in Solidity code and its logic.

The service to handle adding or removing active DHT bootnodes uses the address of the caller to be able to register as a bootnode, as shown in figure 4.6 below [26]. First the function checks if the caller address can be put in a new array or if it should be at the front of an existing array [20]. This is controlled by checking how much time has passed since someone interacted with the bootnode list. Then if 60 seconds have passed a new array gets created. Otherwise the node is pushed to the front and every third node is rotated out of the list. This distributes the load of handling registrations of new users across all clients in the chat.



```
function modDhtList(string calldata addr) public payable {
    uint checkTtl = block.timestamp - dhtListTtl;

    if (checkTtl > 60) {
        dhtList = new string[](0);
        dhtListTtl = block.timestamp;
        dhtList.push(addr);
    }
    else {
        if (dhtList.length > 2) {
            dhtList.pop();
        }
        dhtList.push(addr);
    }
}
```

Figure 4.6: The DHT bootnode setter to populate a bootnode list which can be used to set up a routing table.

4.4.2 Messaging application

This chapter shows the different services from the client side of the design in chapter 4.4.2 and explains how they are used. These services explained below use the interfaces created when compiling the Solitidy source code in chapter 4.4.1 and are used to interact with the Ethereum full node [43]. The services presented here are also used to set up the peer-to-peer messaging capabilities which are implemented through the Libp2p package to build a decentralized application [36].

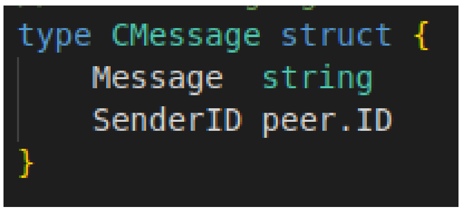
- The routing Service in libp2putils.go package is implemented through Libp2p libraries in Golang [36]. This package is used to create a host interface that can communicate with peer-to-peer protocols and a routing service using the Kademlia DHT routing protocol [26].

  First the NewHost function creates a Libp2p host which gives a user the capabilities to join the peer-to-peer network [36]. The function first uses a seed value to create a unique private key which is used to generate a users content identifier or sign the users messages [24]. The next part of the function creates a multi-address by using the content identifier combined with a TCP listening port [36]. This multi-address is used to create a routable address in the peer-to-peer network, which points to the original caller's address. It also uses a security framework called noise for key exchange and encryption of the traffic instead of TLS [45][46]. This is because noise is more lightweight since it only uses an elliptic curve algorithm for asymmetric encryption used for key exchange. The keys exchanged are generated by using the ChaChaPoly encryption algorithm to generate the symmetric encryption keys used for encrypting traffic between peers.

Then there's the NewDht function which in iteration one returned a DHT table that was initialized by a separate function, which ran in the background [26]. This was then removed since the Pubsub discovery mechanism could be directly populated by the DHT table, as long as it was returned as a discovery service and then given as an argument to the Pubsub service. This eliminated a lot of CPU overhead since one CPU thread less was needed when running the application.

The new implementation instead takes a context as an argument, which is used to initialize the DHT routing table and a host interface [26]. It also takes a list of boot peers as an argument that can be used in the options list. This function also takes arguments to interact with the Ethereum full node. Then it creates the options list with the user inputted variables and sets the node to server mode so it can act as a bootstrap node. It also disables the default bootnode list so the function won't make unnecessary network connections. Lastly a function that can be called whenever the routing table is empty is added to the options list. This function, which populates the routing table, uses the Ethereum service to get the array of active bootnodes. Then the DHT routing table can be returned and be used by a host to continue populating the routing table.

- Messaging service in pubSubService.go package first uses a type structure to represent the message data and the content identifier to map the sender to a specific message as shown below [47].



```go
type CMessage struct {
	Message  string
	SenderID peer.ID
}
```

Figure 4.7: The figure shows the structure for messages where the content id is representedd by variable SenderID and the message is in string format.

The second structure used by the Message service contains vital data for subscriptions to different topics as presented below in figure 4.8. For example, the Message channel is used to store incoming messages or a context used to keep track of messages received by the application [47].

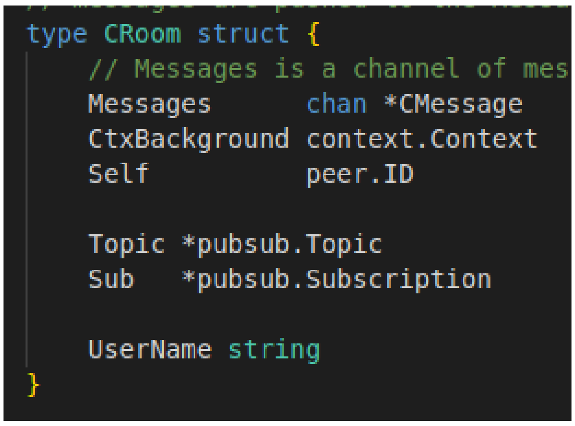

```
type CRoom struct {
    // Messages is a channel of mes
    Messages      chan *CMessage
    CtxBackground context.Context
    Self          peer.ID

    Topic *pubsub.Topic
    Sub   *pubsub.Subscription

    UserName string
}
```

Figure 4.8: The variable CRoom structure used by the messaging service to keep track of an active chat.

The message service is first initialized by using the JoinChatRoom function. The JoinChatRoom function takes arguments to create a topic with the room name that can be used to send messages. Then through this topic a subscription is created, which is used to receive messages [47]. These values are then used to initialize a structure to keep track of a session. The session can then be used to execute the messageLoop function through a thread which can receive messages concurrently from other users.

- The Client Ethereum service in ethereumService.go package is used to handle interaction with the Ethereum full node. By creating or reading a keystores private key based on the implementation from the book on Ethereum go development [43]. This keystore is what represents a wallet for the messaging application and is used to identify the user when interacting with the blockchain. The other functions created by this service are the getters and setters. These functions use transaction data as shown in figure 4.9. Here the interesting values are the GasLimit and GasPrice which in this case are set to 0 to get a default price set by the Ethereum node. The From option sets an address of a user owning this specific transaction. This can then be used when interacting with their own information through their wallet at a later time. These options can then be used when interacting with the smart contract, as shown in figure 4.9.

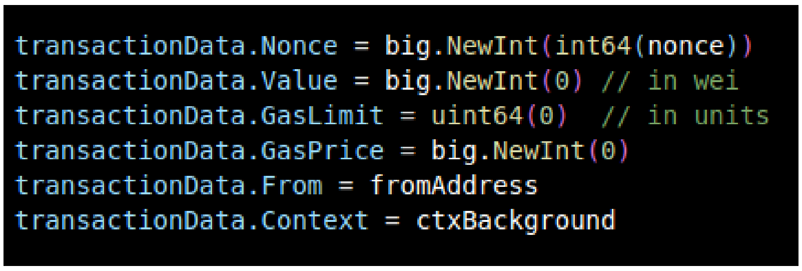
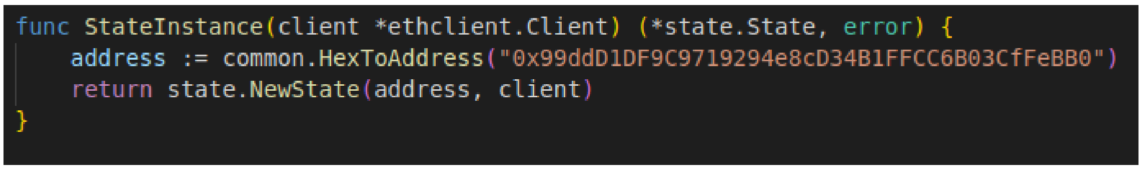

```
transactionData.Nonce = big.NewInt(int64(nonce))
transactionData.Value = big.NewInt(0) // in wei
transactionData.GasLimit = uint64(0)  // in units
transactionData.GasPrice = big.NewInt(0)
transactionData.From = fromAddress
transactionData.Context = ctxBackground
```

Figure 4.9: Structure for transaction data needed to execute a write function against a smart contract on the Ethereum blockchain.

The getters and setters functions also interact with the smart contract through a function called StateInstance [20]. This function uses an Ethereum client as argument, which is used to create a connection to a specific smart contract through a hexadecimal address as shown in figure 4.10.



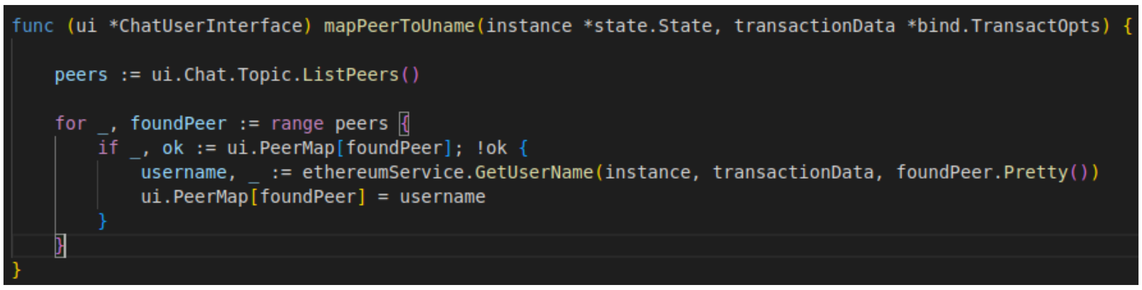
```go
func StateInstance(client *ethclient.Client) (*state.State, error) {
    address := common.HexToAddress("0x99ddD1DF9C9719294e8cD34B1FFCC6B03CfFeBB0")
    return state.NewState(address, client)
}
```

Figure 4.10: Function to connect to a specific smart contract through the Ethereum client function and returning its state.

These are implemented by using the interfaces that gets created when compiling the smart contract through Solidity [43]. For example, these functions are used to register a user, verify a user, get a DHT bootnode list or modify the bootnode list according to the design in figure 4.3.

- The UI service in userInterface.go package is implemented according to the application go-libp2p-pubsub chat example and uses the code in the ui.go as a skeleton with some changes to fit this implementation [48]. It uses an input channel variable to read data from the users STDIN and a map variable that maps peer ids to usernames through a hashmap which was implemented to fit this application's needs.

  The Run function is changed to take an Ethereum state and Ethereum client to be used by the modified EventHandler function. The modified EventHandler implementation uses the built in Libp2p Pubsub function to publish directly to the topic and uses the input from the user to do an explicit type conversion from a string to an array of bytes [47]. The EventHandler also uses the mapPeerToUname to populate the hashmap with peer id as a key and username as a value. The mappPeerToUname function basically first gets all peer id in a chat through the Pubsub built in function to list all peers that are part of a topic. Then it iterates over all peer ids found and if the hashmap does not already contain a peer id, it will add it by querying the Ethereum service for the username. Then the returned value can be stored in the hashmap by pointing the key peer id to the value of the username as shown below.

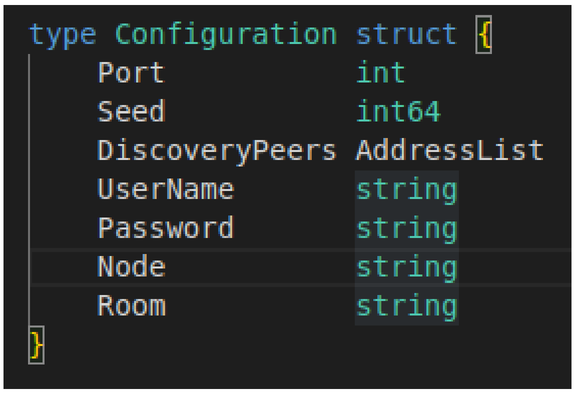


```go
func (ui *ChatUserInterface) mapPeerToUname(instance *state.State, transactionData *bind.TransactOpts) {

    peers := ui.Chat.Topic.ListPeers()

    for _, foundPeer := range peers {
        if _, ok := ui.PeerMap[foundPeer]; !ok {
            username, _ := ethereumService.GetUserName(instance, transactionData, foundPeer.Pretty())
            ui.PeerMap[foundPeer] = username
        }
    }
}
```

Figure 4.11: This shows the mapPeerToUname function that is used to populate a hashmap with new users which are mapped through the content identifier to a username.

Then after the hashmap has been updated, a function called refreshPeers is used to clear the UI and print the usernames from the hashmap to the terminal.

- The program in the main.go package binds everything together and is executed from the command line. It takes either zero or more input flags from STDIN. If zero input flags are given then default values are used. These flags are set through a helper package called Configuration and it is structured as shown below in figure 4.12.

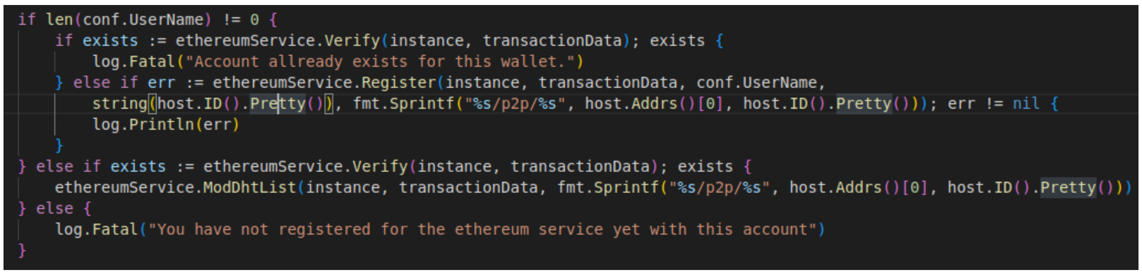


```go
type Configuration struct {
	Port           int
	Seed           int64
	DiscoveryPeers AddressList
	UserName       string
	Password       string
	Node           string
	Room           string
}
```

Figure 4.12: Configuration package with structure used to set up a chat session and to store input from a user given through STDIN.

When running the application, the configuration package is used to set the username and seed values. An Ethereum client is used to create the connection to an Ethereum smart contract [43]. A password is used either to create or to open an Ethereum wallet using the Ethereum service package to gain a private encryption key. The key is then used together with a context and an Ethereum client to create the Transaction data needed to interact with the smart contracts services.

Then a Libp2p host is set up through the routing service package using the seed and port variables to publish itself on the network as an active Libp2p node [36]. The newly generated Libp2p host can then register itself by using its Ethereum wallet [43]. When the node registers for the first time it will set itself up as an active DHT boot node to gain routing capabilities in the peer-to-peer network through the Registry service using the Transaction data [26][43]. If the Ethereum wallet is already registered then it only sets up the routing service by calling the function ModDhtList as shown in figure 4.13.

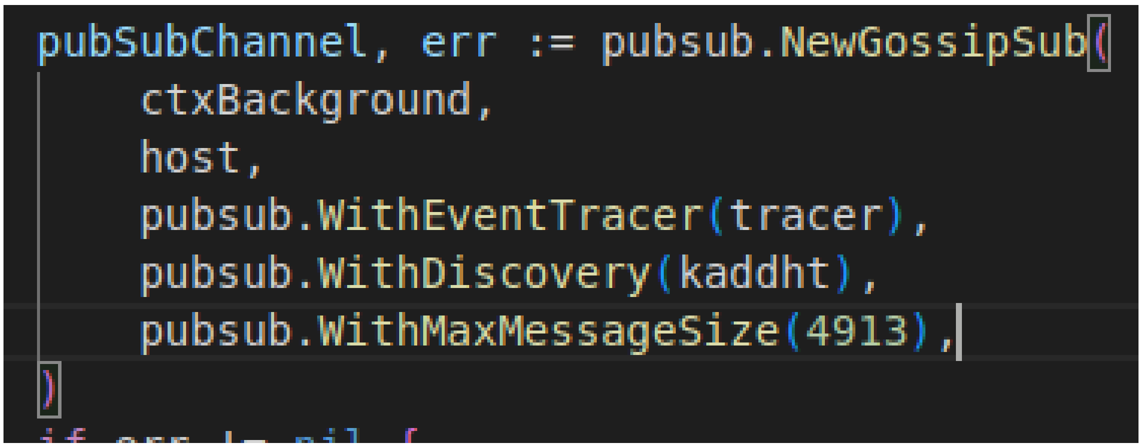


```go
if len(conf.UserName) != 0 {
    if exists := ethereumService.Verify(instance, transactionData); exists {
        log.Fatal("Account allready exists for this wallet.")
    } else if err := ethereumService.Register(instance, transactionData, conf.UserName,
        string(host.ID().Pretty()), fmt.Sprintf("%s/p2p/%s", host.Addrs()[0], host.ID().Pretty())); err != nil {
        log.Println(err)
    }
} else if exists := ethereumService.Verify(instance, transactionData); exists {
    ethereumService.ModDhtList(instance, transactionData, fmt.Sprintf("%s/p2p/%s", host.Addrs()[0], host.ID().Pretty()))
} else {
    log.Fatal("You have not registered for the ethereum service yet with this account")
}
```

Figure 4.13: This shows the main functions registration and validation process of a new or old user.

After the account is registered or added to the boot node list, the main function will query the Ethereum node for all active bootnodes in the peer-to-peer network and store these in a list [43]. The list can then be used to set up the nodes routing capabilities through the routing service.

The Gossip router can be set up by giving arguments that are used to listen for incoming messages through the host interface, as shown below in figure 4.14 [47]. The tracer is used for logging capabilities and the routing service is added through the WithDiscovery flag. The last flag is used to force a max message size to follow the LoRaMesh constraints found in chapter 2.3.



```go
pubSubChannel, err := pubsub.NewGossipSub(
    ctxBackground,
    host,
    pubsub.WithEventTracer(tracer),
    pubsub.WithDiscovery(kaddht),
    pubsub.WithMaxMessageSize(4913),
)
```

Figure 4.14: This shows the setting up of a gossip router with the routing service, tracer and a maximum message size.

The gossip router is then used by the message service JoinChatRoom function to set up a publish and subscribe service [47]. The returned value from JoinChatRoom is then used to initialize the user interface (UI) service and then start the chat UI through the Run function.

4.4.3 Test implementation

This chapter implements a testsuite through the testutils.go package that is used to wrap the main function in a runtime environment using Testground or to gain trace logs capabilities for the proof-of-concept application [47][40].

First when using Testgrounds runtime environment a random username and seed is generated using a generator [40]. The environment also initializes a network environment by using a MaxMessageSize variable set to 4913 bit/s to simulate the IoT network constraints which were discovered in chapter 2.3. This network environment is also used to simulate an IPv4 network for all simulated nodes.

The UI service implemented in userInterface.go is also changed for the testsuite. The change is done by adding the testEventHandler function that mimics the original function eventHandler. Except that messages are sent every 1 seconds with a fixed value of 64 bytes size. Using a counter to keep track of messages sent. When it has sent all 60 messages the counter will be equal or higher to 60 and then exit automatically. This will guarantee that it publishes all messages in a test run. The testEventHandler sends a done value to close the message channel and returns to the main function. The runtime environment will signal that it has successfully terminated by returning a nil value to the calling function [40]. This signals to Testground that it has successfully finished executing the test for a node.

5 Results

The results chapter shows results from two iterations that was executed in the simulated environment called Testground and shows the fourth step demonstration of Peffer et al. design science method [40][13]. Each iteration was a representation of the Design and development, Demonstration and Evaluation steps of Peffer et al.s implementation of the Design science method, see chapter 3.1. For each iteration, test runs were made in a controlled experiment using independent variables in the form of nodes to see how it affected the dependent variables of the messages network propagation delay [14]. The bandwidth was limited to 4913 bitrate/s, each message had a length of 64 bytes. One message per second was sent until 60 messages had been sent.

Iteration one used a sample size of up to 15-105 simulated nodes spread into 7 different test groups. Iteration one's results gave low network propagation delay using 15 nodes and a significant incretion in network propagation delay for 30 nodes. This is because to many nodes are directly connected, see chapter 4.3 for further explanation. Iteration two will test if it also holds for 10 nodes when these are forced to be directly connected. Iteration two used a sample size of 10-105 simulated nodes in a network spread into 13 different test groups. Where iteration one executes each test group 4 times with a total of 28 test runs. Iteration two ran each test group for 31 test runs with a total of 403 test runs.

Iteration one is presented below in table 5.2 and contains the raw data from all its test runs. Followed by the data from iteration two that are visually presented here through graphs, histograms and is derived from raw data that can be found on GitHub together with analysis code created through the R programming language in B Appendix 1.

5.1 Iteration one

Iteration one is presented first in table form to present the raw data and is analyzed in chapter 6 evaluation.

Test run	Nodes	Published Messages	Total Delivered Messages	Average network Propagation delay (ms)
1	15	885	13355	3
2	15	885	13300	2
3	15	885	13298	3
4	15	885	13331	4
5	30	1771	53192	207
6	30	1771	53160	240
7	30	1770	53184	215
8	30	1770	53208	170
9	45	2656	119781	200
10	45	2655	119752	341
11	45	2655	19689	236
12	45	2655	119809	366
13	60	3542	212819	389
14	60	3543	212805	431
15	60	3542	212565	442
16	60	3541	212905	393
17	75	4433	332031	973
18	75	4430	331730	880
19	75	4427	331945	934
20	75	4430	331900	868
21	90	5316	474464	2404
22	90	5314	475343	2456
23	90	5314	475949	2374
24	90	5321	475488	2494
25	105	4641	469237	5276
26	105	4711	477385	5022
27	105	4628	463405	5755
28	105	4413	443909	5474

Table 5.2: Chosen results from 7 different test groups. Showing published messages, delivered messages and propagation delay measured in ms.

5.2 Iteration two

This iteration shows the results from all 403 test runs represented through histograms and graphs. Created from raw data and R source code that can be found on GitHub, see link in B Appendix 1.

The Histogram below in figure 5.16 was used to measure the extraneous variables published messages and delivered messages. These variables was used to determine if the results were valid and could be used for further analysis, see chapter 3.4 for further information of extaneous variables. Figure 5.16 below shows a histogram where the Y-axis shows the number of test runs and the X-axis shows the percentage of successfully delivered messages for those test runs.

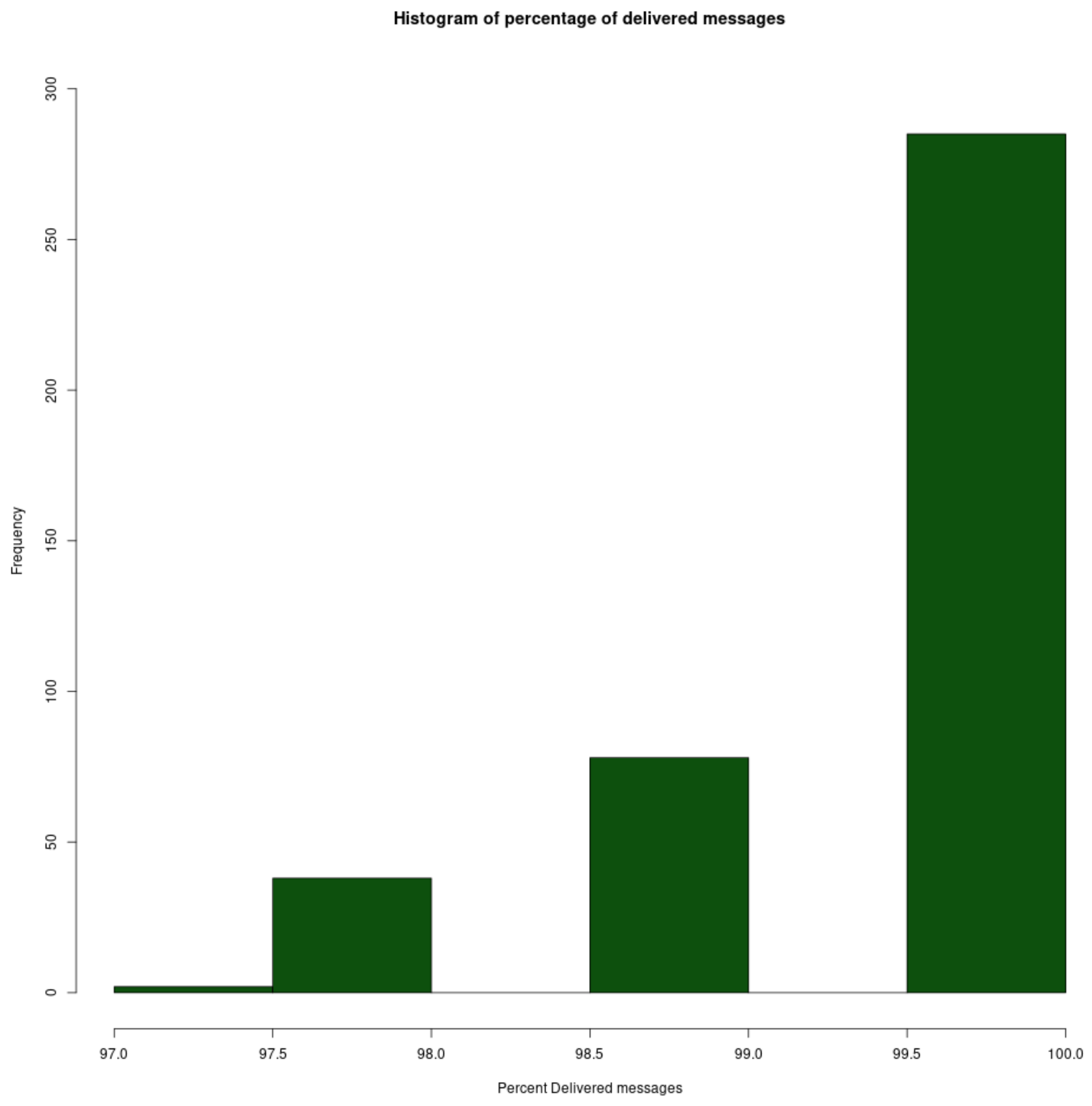


Figure 5.15: The test results for the extraneous variables are presented through a histogram showing the percentage of delivered messages distributed over all 403 test runs.

The histogram below in figure 5.16 shows the network propagation delay spread and is analyzed in chapter 6.2 to see if it follows a bell curve, see chapter 3.4 for further explanation on a bell curve.

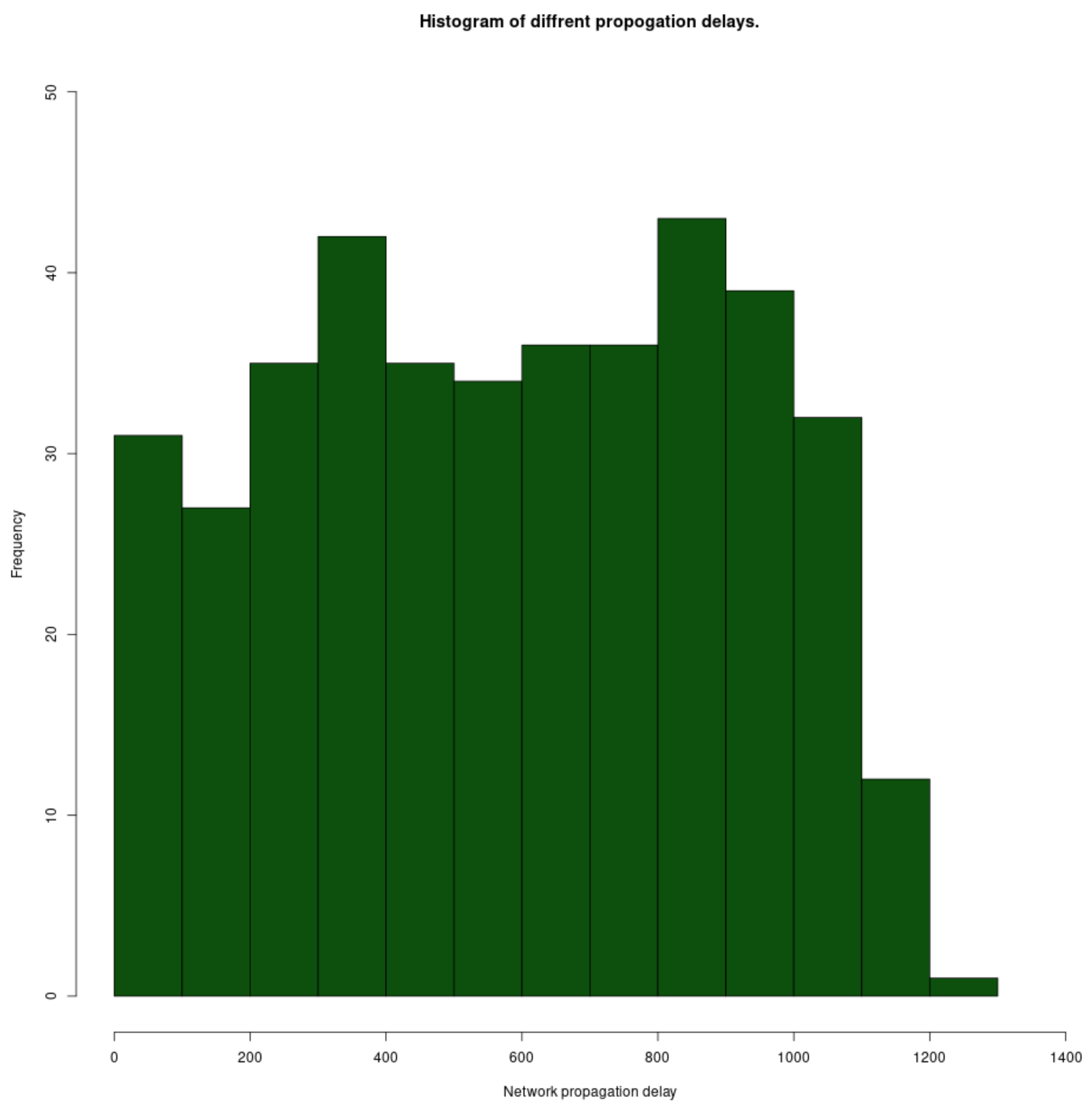


Figure 5.16: This figure shows a histogram of a count of all different network propagation delays which were found spread across the test runs.

The figure 5.17 shows a scatter plot which is used to get a visual overview of the data to determine if linear regression can be applied.

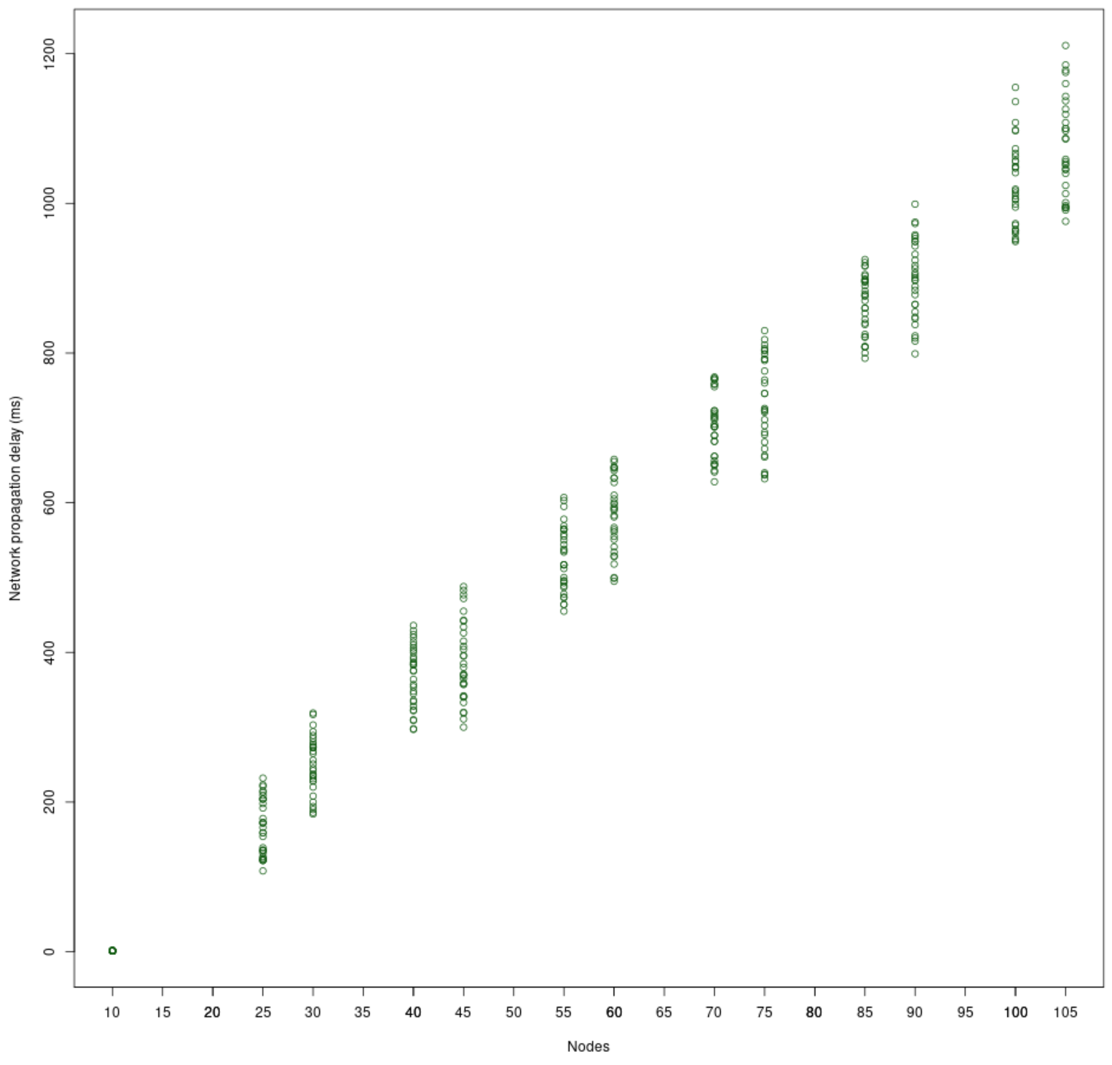


Figure 5.17: Shows the means of all sent messages network propagation delay through a scatter plot for all 403 test runs.

The next part of the results shows the residuals which summarizes the error between the predictions in the model and the actual results in figure 5.18 [31]. These are used to calculate the variance and outliers in the results as shown in figure 5.18.

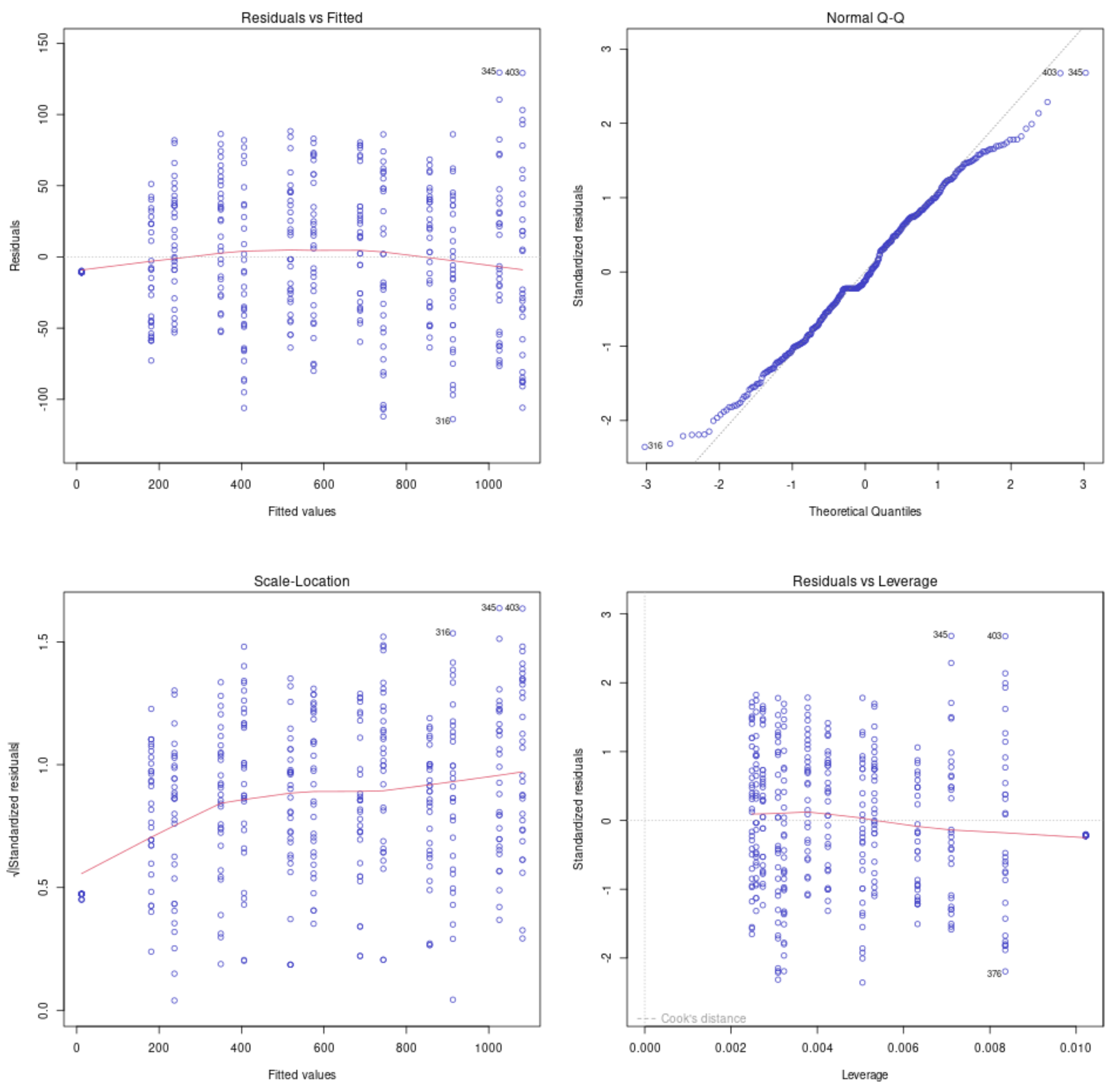


Figure 5.18: This figure shows four different graphs where it shows the unexplained variance and if there exists any extreme variables called outliners in the test results.

- Where the first graph in figure 5.18 called Residual vs Fitted and shows if there is a linear relationship between the data points in the model [49]. Residual vs Fitted should have the line be as horizontal as possible for this assumption to be fulfilled.
- Next in figure 5.18 follows the Normal Q-Q graph and this controls normality distribution in the residuals assumption of the model [49]. The Normal Q-Q should have the majority of the residual follow the dashed line for the assumption to be fulfilled.
- Then in figure 5.18 the Scale-Location graph shows the homoscedasticity of equal variance in the residuals [49]. The Scale-Location shows if the residual is spread randomly by having a horizontal red line.

- The last graph in figure 5.18 called residuals vs leverage shows all influential values which could be removed to change the results of the test [49].

6 Analysis

This chapter shows the fifth step evaluation of Peffer et al. design science method [13]. The analysis is divided into iteration one and iteration two analysis which evaluates the results from chapter 5.1 and chapter 5.2. Chapter 6.1 only interpreted the data that was either erroneous or did not fit the model of linear regression from previous iterations. Then in iteration two descriptive statistical and inferential statistical analysis is applied to the end data to interpret and analyze the results from the results chapter 5.2 of iteration two [31][15].

In iteration two the test data is evaluated by interpreting the results from the histogram showing the percentage of delivered messages distributed over all 403 test runs from chapter 5.2. Then it evaluates if linear regression can be used on the results by controlling that the test data follows a normal distribution, that the results are linear and that the model follows the assumption of homoscedasticity, see chapter 3.4 for further explanation [31]. Then results are analyzed through an inferential statistical analysis method through linear regression that will help to interpret the independent variable and dependent variables relationship. Through an estimate effect on the regression model which helps with seeing the cause-and-effect relationship exists. P-value results to see if null hypothesis can be rejected, t statistics which shows if the results occurred by chance, standard error to see the variation between the independent and dependent variables.

6.1 Iteration one analysis

From table 6.3 it can be seen that not all messages were successfully published every run which creates an internal validity issue in the test results since the results from the test run can not be correctly measured [34]. For example, 855 messages for 15 nodes is too low since 15 nodes sending 60 messages each would at least have published 900 messages. Also delivered messages were so low one time that the measured data can't be included in a valid test run since the percentage of total delivered messages were so low as shown below where part of table 6.3 is included.

Test run	Nodes	Published Messages	Total Delivered Messages	Average network Propagation delay (ms)
11	45	2655	19689	236

Table 6.3: Showing erroneous results from a test run with the published messages, delivered messages and propagation delay measured in ms.

The percentage of delivered messages for run 11 would be :

$$19689 \,/\, (45 * 2655) * 100$$

Which would equal 16 percent delivered messages. This shows that the dependent variable Network propagation delay relies on the results of messages being delivered. Since the Network propagation delay would show an erroneous results if only calculated on 16 percent of the delivered messages which would be an internal validity issue [34].

The next issue is from chapter 5.1 in table 5.2 where all test runs containing 90 nodes publish more messages than all test runs containing 105 nodes. This could indicate an application problem or design problem where the application can't handle the load of publishing and receiving that amount of messages at the same time.

The last issue can be found in the scatter plot below from figure 6.19 which is derived from the raw data found in table 5.2 from iteration one. The data points of the scatter plot shows that the data is clearly not linear and seem to have a large network propagation delay increase between each test run and cannot be explained by the model. For example, using 15 nodes seems to have almost no delay effect on the population, but increasing to 30 nodes has a significant effect on the network propagation delay as can be seen in figure 6.19

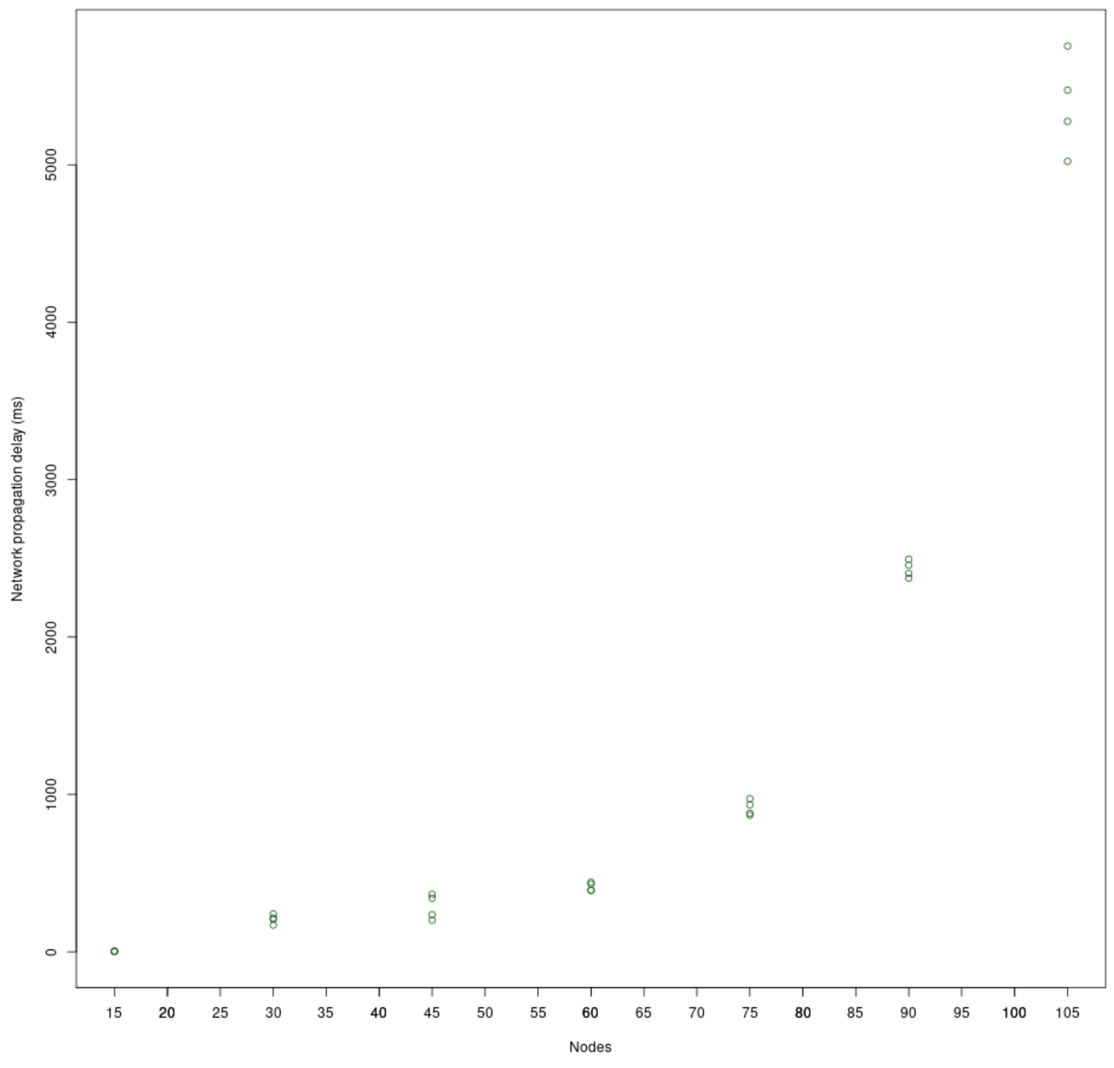


Figure 6.19: Shows the means of each run's network propagation delay through a scatter plot for all sent messages.

The conclusion that can be drawn from the first iteration is that changes to both the design and the application is needed. Where changes to how the application utilizes the

CPU could help the issue of not publishing enough messages each test run and not getting enough delivered messages.

The next issue with the significant increase in network propagation delay between the groups could be because only one DHT bootnode was used which is a design problem [26]. There was also a significant increase of delay between 15 nodes and 30 nodes shows that changes to the test interval could be needed. A conclusion here is to see if there is the same network propagation delay for 10 nodes and then apply more frequent node distribution in the test interval so see if the results follow a pattern.

The issue with the extremely low percentage of delivered messages indicated a problem of receiving messages that was discovered to give rise to unreliable results in the measured data. This could show that there exists extraneous variables that need to be controlled in order to trust the results from the data that is measured [32].

6.2 Linear regression analysis

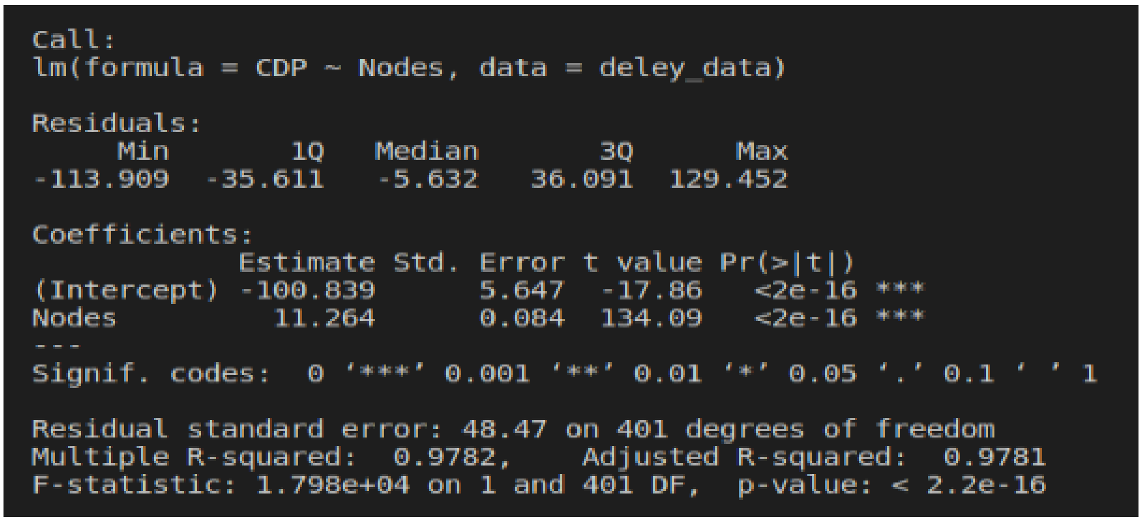
The results in chapter 5, figure 5.15 shows a histogram of percentage of delivered messages. It indicates a successful test execution based on the results of the lowest amount of delivered messages being 97 percent.

The histogram in chapter 5, figure 5.16 is showing all different network propagation delays across all test runs from iteration two. It shows that the highest network propagation delay is 1200 ms which is a significant improvement from iteration one after the changes on the design and proof-of-concept application been applied. This histogram shows that there is a normal distribution of the test results since it follows a bell curve with small peaks distributed across it which is acceptable [31].

The scatter plot in chapter 5, in figure 5.17 shows a linear distribution of the results across the test groups with the values of 10 nodes having the same delay as 15 nodes from iteration one [31]. This could indicate that below a certain number of nodes there is no significant increase in network delay which can be attributed to having a direct peering agreement between more nodes and not having to relay messages between several nodes to reach a destination [38][35].

Lastly the homoscedasticity assumption is controlled. The assumption states that residual unexplained variance should not change significantly over the range of predictions in the model [31]. This is controlled in that the prediction error does not change significantly across the range in the prediction model. This is done by controlling the four graphs in chapter 5, figure 5.18 [49]. The first graph called residual vs fitted shows if there is a linear relationship between the data points by being horizontal close to zero. Normal Q-Q checks for normality distribution in the residuals assumption of the model and if a high amount of residuals follow the assumption is fulfilled. Then the Scale-Location checks for homoscedasticity of equal variance in residuals and checks if the data is randomly spread if the line is horizontal. This shows that the first values from the 10 runs are not equally randomly spread which gives a slight curve in the beginning. This result is acceptable and since it can be explained from the fact that with a direct peering agreement between nodes there is almost no network propagation delay between messages. The last graph called residuals vs leverage shows all influential values which could be removed to change the results of the test and is not of interest since the model seems to fulfill the assumption of homoscedasticity for the most part which was all that is needed to apply linear regression analysis on the results.

To apply the linear regression model the programming language R is used with the lm function which takes a given formula to create the model [31]. All different parameters used to create the model and the results are shown below in figure 6.20.



```
Call:
lm(formula = CDP ~ Nodes, data = deley_data)

Residuals:
     Min       1Q   Median       3Q      Max
-113.909  -35.611   -5.632   36.091  129.452

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -100.839      5.647  -17.86   <2e-16 ***
Nodes         11.264      0.084  134.09   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 48.47 on 401 degrees of freedom
Multiple R-squared:  0.9782,    Adjusted R-squared:  0.9781
F-statistic: 1.798e+04 on 1 and 401 DF,  p-value: < 2.2e-16
```

Figure 6.20: This shows the summary of the linear regression model applied to the raw data from iteration two on the nodes and network propagation delay.

The interesting sections here are that the residuals which summarizes the error between the predictions in the model and the actual results [31]. These are used to calculate the variance and outliers in the results which was discussed earlier in this chapter when checking that the data follows the assumption of homoscedasticity through the results from figure 5.18.

The Coefficients section from figure 6.20 in which the first two values of the Estimate give the Y and X of the linear regression model which in this case are -100.839 ms for Y and 11.264 for X [31]. This describes that the effect of Nodes on network propagation delay is 11.264 ms.

The standard error from figure 6.20 which is 5.647 ms for the network propagation delay and 0.084 for the nodes [31]. These values are used to calculate the T values that can be seen in figure 6.20. The T value shows how close the distribution is to the null hypothesis. The further it is from zero the better since it then shows that the results did not occur by chance. This is then followed by the P-value of the Coefficients section that shows if there exists a significant relationship with the null hypothesis. It shows the chance of seeing the effect between nodes and network propagation delay if the null hypothesis would be true. The values which are seen here showed that the t-values are not zero and that the P-value is really low with a value of 0.0000000000000002 which indicates that the null hypothesis can be rejected.

The last line of figure 6.20 shows a P-Value of 0.0000000000000002 [31]. This indicates that the data fits the model well and is presented through the graph below.

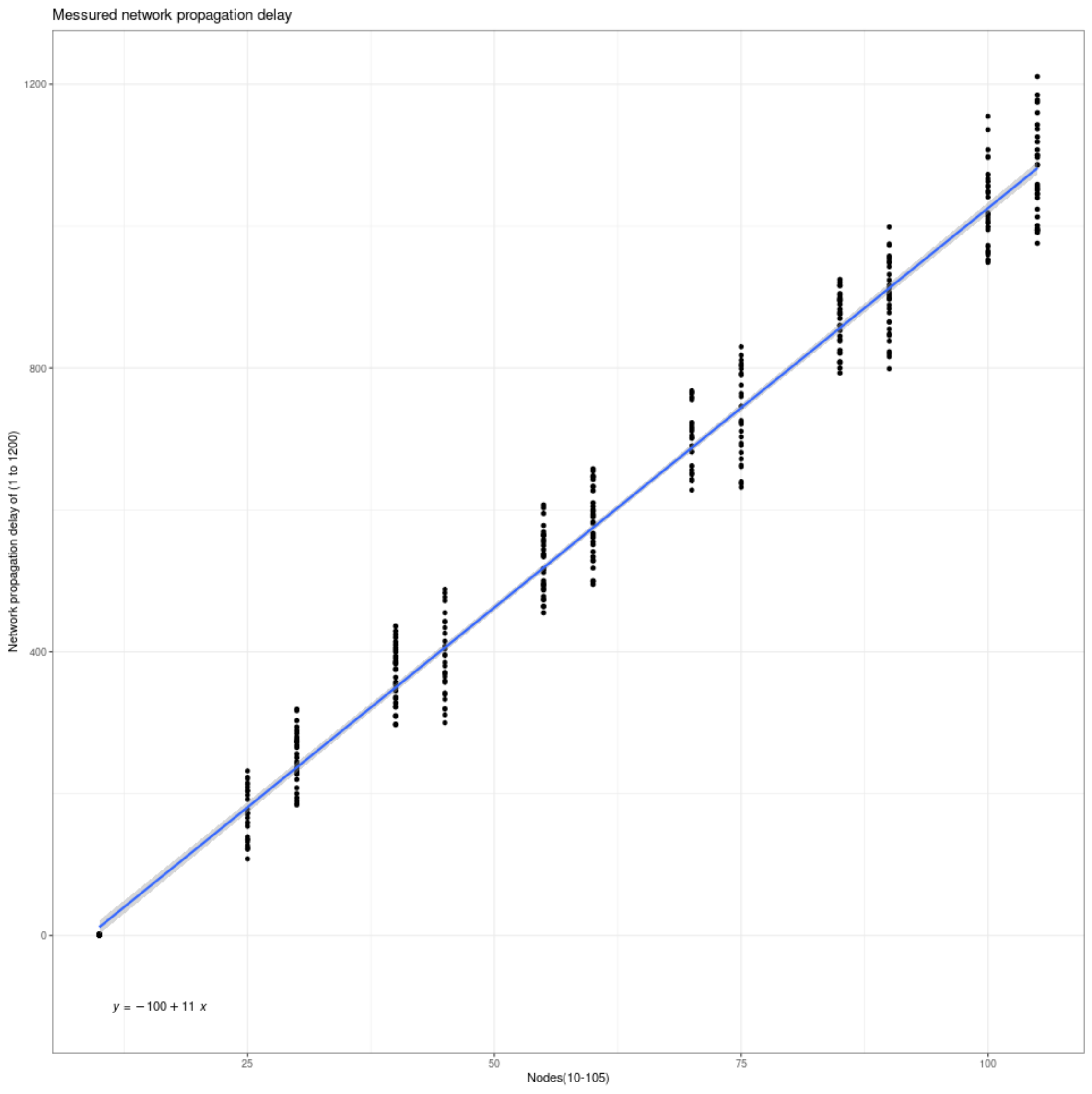


Figure 6.21: The figure shows the scatter plot with a regression line to show the cause-and-effect relationship between the independent variables of Nodes by applying the value of -100.839 for Y and 11.264 for X.

Figure 6.21 above compared to figure 6.19 that had a high increase in network propagation delay between the test groups shows that by removing the DHT node bottleneck the network propagation delay got lower and the results became more linear which made it possible to measure the data more accurately. Figure 6.21 also shows that there exists a significant positive relationship between Nodes and network propagation delay with a P-value less than 0.05 and a 11.264 ms increase in delay for every node increase. This also indicates that the null hypothesis can be rejected since there is a clear cause-and-effect relationship between the amount of nodes and network propagation delay between messages [30].

7 Discussion

This thesis implements the theoretical suggestions by De Klerk et al. Where he suggests using blockchain technologies together with peer-to-peer technologies to complement the weaknesses of IoT networks [5]. It is implemented by first creating a design based on knowledge acquired on the LoRaMesh protocol, Ethereum blockchain and the Libp2p protocol. The design is then used to create a proof-of-concept application that is then tested through a controlled experiment to answer the research questions. Results from the controlled experiment are then discussed in this chapter together with the results from the linear regression analysis and the limitations of this implementation.

The results from the controlled experiment shows that this application can, in a simulated environment, handle communicating with an Ethereum node receiving and sending 1 message every second with the size of 64 bytes with a bandwidth of 4913 bits/s for up to 105 nodes. The mean of all sent and received messages for the network propagation delay did not get higher than 1200 ms when having 105 nodes in the simulated environment. Then these results from iteration two for measuring the cause-and-effect relationship between nodes and network propagation delay could be used because they contain high internal validity and has been thoroughly tested through linear regression analysis [31]. The results are also reliable since they could be reproduced by setting up the controlled experiment, see B Appendix 1. Results from the linear regression show that the null hypothesis can successfully be rejected since the P-Value is lower than the chosen P-value of 0.05. These results then show that the message application can perform well under high pressure and in turn validates that the Design and Proof-of-concept application can be used to answer research questions RQ1 and RQ2 as discussed below.

Research question RQ1 asks how can blockchain be implemented in an IoT network to build a decentralized solution which is used to find people to communicate with? RQ1 could be answered through a design and a proof-of-concept application that was created by applying a design science method. It needed two iterations to obtain a result that indicates how the solution should be implemented to handle weaknesses of IoT networks, as suggested by De Klerk et al [5]. The weakness of protocols like DNS could be handled by implementing a blockchain node to store information through smart contracts. Where in iteration one a seperate DHT bootnode was used to keep track of all the users joining the network and the smart contracts where only used for registration purposes. But in iteration two the smart contract was used to keep track of users who joined the network and distribute routing information to users who joined by distributing a bootnode list that used Kademlia DHT algorithm for routing and discovering new peers [26]. Together with a peer-to-peer protocol called Gossip, implemented through Golang Libp2p library to facilitate sending and receiving messages between users [36].

Next research question RQ2 asks what technologies are needed to implement a blockchain solution in an IoT network to facilitate an emergency communication network when considering the limitations on network bandwidth? RQ2 could be answered by implementing blockchain nodes on a separate full node which could handle the computer load as was done in Alsayed Kassem et al. research [12]. In his research they also use a smart contract with a registration service which gives users the opportunity of controlling the data they load which was also done by this application but together with a messaging application. Then together with the Proof of authority algorithm as suggested by De Klerk et al. study where they suggest not implementing a Proof of work algorithm to counteract high network and computer loads for blockchain technologies [5].

The implementation had some limitations in the form of how the controlled experiment was implemented and the generalizability of the results. The controlled experiment was implemented by using Testground to simulate nodes [40]. This gave a lot of overhead for each test in the form of unnecessary functionality which could have affected the results in the form of network overhead and should be considered when evaluating the results.

Findings from the experiment are generalizable for a peer-to-peer network using the Gossip protocol in a controlled environment. Since it relies on the default setting of the gossip protocol when choosing its peering degree between directly connected peers [38]. This made the results more generalizable in the context of a peer-to-peer network. The results are generalizable in this context since it actually had to rely on the Gossip protocol's peer-to-peer capabilities when distributing a message to a peer in the network. The findings also show that blockchain smart contract can be used to distributed a DHT bootnode table to achieve a distributed emergency communication message application for LoRaMesh under these network constraints in the controlled environment[26][28]. This will help protect against a weakness in the IoT network, as suggested by De Klerk et al. by having a a distributed infrastructure which keeps all the necessary information to achieve routing capabilities in the emergency communication network [5]. Though it will not be generalizable outside this controlled environment since it only tests how well the application can perform under IoT network constraints that exist for the LoRaMesh protocol [28].

8 Conclusions and Future Work

In today's society we need an emergency communication system to facilitate communication for when disaster strikes. Where the previous attempts only focused on the network communication and were missing a messaging capability between users. This gave rise to the central research questions RQ1 and RQ2 presented below.

RQ1: How can blockchain be implemented in an IoT network to build a decentralized solution which is used to find people to communicate with?

RQ2: What technologies are needed to implement a blockchain solution in an IoT network to facilitate an emergency communication network when considering the limitations on network bandwidth?

To answer these questions a design was created based on LoRaMesh, Ethereum blockchain technologies and peer-to-peer protocols. This was then implemented through a proof-of-concept application which could then be tested through a controlled experiment to validate how it performs under high pressure to control if RQ1 and RQ2 could be answered with the proposed solution.

RQ1 could be answered since the application was successfully implemented in a decentralized manner by using the Kademlia DHT protocol to find peers in the peer-to-peer network. This was done by using the described solution shown below:

- By using blockchain smart contracts to distributed DHT bootnode list and only distribute the new node which joined the peer-to-peer network.
- By using DHT protocol combined with the Libp2p library which was used to implement the gossip protocol. The gossip protocol then implemented the DHT protocol to archive the routing functionality for the messaging application.

RQ2 was answered by the controlled experiment which showed that this application can, in a simulated environment, handle communicating with an Ethereum node receiving and sending messages under a limited bandwidth of 4913 bit/s. Where the mean of all sent and received messages network propagation delay did not get higher than 1200 ms when having 105 nodes in the simulated environment. It was achieved by using the technologies shown below :

- By using DHT protocol to facilitate finding peers and gain a decentralized solution.
- By using Ethereum node with the proof of authority algorithm for validation on a separate node.
- By using Libp2p library through Golang to implement the Gossip protocol to gain a messaging functionality.

The findings from the controlled experiment shows there is a positive relationship between the amount of nodes and the amount of network propagation delay for the messages. By using a linear regression analysis a cause-and-effect relationship showed that the null hypothesis can be rejected and through this the alternate hypothesis could be confirmed to hold true. This gives rise to the conclusion that the message application could be a solid solution for an emergency communication network. I believe that the design and proof-of-concept application could be used to further build on an Emergency communication network for civilians.

The results from this thesis helped contribute to new knowledge about how to distribute bootstrap nodes in a Kademlia DHT network. Since to the best of my knowledge there exists no good way to distribute bootstrap nodes except statically or through a centralized location. This thesis solution on how to distribute the bootstrap nodes through a smart contract which was added after iteration one can further the peer-to-peer network in decentralizing its infrastructure. Though further investigation is also needed to check the indication, that below a certain number of nodes there is no significant increase in network delay. This behavior could be attributed to having a direct peering agreement between more nodes and not having to relay messages between several nodes to reach a destination, which was found out from iteration one and two.

Future work based on this thesis would be distributing the Ethereum full nodes network addresses through an IPFS network and adding the capabilities to the clients so that the IPFS part would also be decentralized. Another part would have been to handle the weakness in an IoT network pertaining to security issues which this application did not test. It should be validated if this solution can help combat the security weaknesses in an IoT network by testing the application in the context of security and implementing best practices design patterns to avoid security vulnerabilities. Another step would have been to specifically test the implementation of the smart contract in a simulated environment where security, logic and performance was measured and evaluated. Then developing a more complete solution for an IoT network for emergency communication which then implements this application. Lastly everything should be tested through a live experiment with real users and real IoT nodes as infrastructure.

References

[1] MSB. (2021) Rakel. [Online]. Available: https://www.msb.se/sv/verktyg--tjanster/rakel/

[2] K. C. V. G. Macaraeg, C. A. G. Hilario, and C. D. C. Ambatali, “Lora-based mesh network for off-grid emergency communications,” 2020. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9342944

[3] E. Andersen, T. Blaalid, H. Engstad, S. Røkenes, and F. T. Johnsen, “A lora mesh network asset tracking prototype,” 2020.

[4] R. Pueyo Centelles, R. Meseguer, F. Freitag, L. Navarro, S. F. Ochoa, and R. M. Santos, “Loramoto: A communication system to provide safety awareness among civilians after an earthquake,” *Future Generation Computer Systems*, vol. 115, pp. 150–170, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X20306063

[5] L. de Klerk, “Blockchain-based dns and pki to solve issues of trust, security and censorship in the context of the iot,” 2021. [Online]. Available: http://resolver.tudelft.nl/uuid:519be644-a7a1-471c-8bfa-8409e409ac75

[6] M. A. Khan and K. Salah, “Iot security: Review, blockchain solutions, and open challenges,” *Future Generation Computer Systems*, vol. 82, pp. 395–411, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17315765

[7] MSB. (2019) Nödtelefoner via rakelnätet ökar tryggheten i samhället. [Online]. Available: https://framtidenskommuner.se/2019/11/11/nodtelefoner-via-rakelnatet-okar-tryggheten-i-samhallet/

[8] C. P. Quitevis and C. D. Ambatali, “Feasibility of an amateur radio transmitter implementation using raspberry pi for a low cost and portable emergency communications device,” 2018.

[9] (2022) Lora. [Online]. Available: https://en.wikipedia.org/wiki/LoRa

[10] (2020) lopy4. [Online]. Available: https://docs.pycom.io/gitbook/assets/specsheets/Pycom_002_Specsheets_LoPy4_v2.pdf

[11] (2021) Proof-of-authority chains - wiki openethereum documentation. [Online]. Available: https://openethereum.github.io/Proof-of-Authority-Chains

[12] J. Alsayed Kassem, S. Sayeed, H. Marco-Gisbert, Z. Pervez, and K. Dahal, “Dns-idm: A blockchain identity management system to secure personal data sharing in a network,” *Applied Sciences*, vol. 9, no. 15, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/15/2953

[13] M. A. R. Ken Peffers, Tuure Tuunanen and S. Chatterjee, “A design science research methodology for information systems research,” 2007.

[14] R. Bevans. (2022) Guide to experimental design | overview, 5 steps and examples. [Online]. Available: https://www.scribbr.com/methodology/experimental-design/

[15] (2022) The beginner's guide to statistical analysis | 5 steps and examples. [Online]. Available: https://www.scribbr.com/category/statistics/

[16] R. Bevans. (2022) Understanding p-values | definition and examples. [Online]. Available: https://www.scribbr.com/statistics/p-value/

[17] T. Badretdinov. (2018) Blockchain explained. [Online]. Available: https://kauri.io/#communities/Getting%20started%20with%20dapp%20development/blockchain-explained/

[18] (2022) Cryptographic nonce. [Online]. Available: https://en.wikipedia.org/wiki/Cryptographic_nonceb

[19] V. Buterin. (2022) Ethereum whitepaper. [Online]. Available: https://ethereum.org/en/whitepaper/#a-next-generation-smart-contract-and-decentralized-application-platform

[20] W. Barnes. (2019) Ethereum 101 - part 5 - the smart contract. [Online]. Available: https://kauri.io/#collections/Ethereum%20101/ethereum-101-part-5-the-smart-contract/

[21] ——. (2019) Ethereum 101 - part 7 - decentralized apps. [Online]. Available: https://kauri.io/#collections/Ethereum%20101/ethereum-101-part-7-decentralized-apps/

[22] ——. (2019) Ethereum 101 - part 7 - the evm. [Online]. Available: https://kauri.io/#collections/Ethereum%20101/ethereum-101-part-7-the-evm/#_top

[23] (2022) Goquorum the following diagram outlines the goquorum high-level architecture. [Online]. Available: https://consensys.net/docs/goquorum//en/latest/concepts/architecture/

[24] (2022) What is ipfs? [Online]. Available: https://docs.ipfs.io/concepts/what-is-ipfs/#decentralization

[25] (2022) How ipfs works? [Online]. Available: https://docs.ipfs.io/concepts/how-ipfs-works/#content-addressing

[26] J. H. raulk raulk and M. Inden. (2021) libp2p kademlia dht specification. [Online]. Available: https://github.com/libp2p/specs/blob/master/kad-dht/README.md

[27] (2022) go-libp2p-pubsub. [Online]. Available: https://pkg.go.dev/github.com/libp2p/go-libp2p-pubsub#section-readme

[28] K. J. Cotrim JR, "Lorawan mesh networks: A review and classification of multihop communication," 2020. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7435450/

[29] M. Suznjevic and J. Saldana, "Delay limits for real-time services," *IETF draft*, 06 2016. [Online]. Available: https://www.researchgate.net/publication/304674431_Delay_Limits_for_Real-Time_Services

[30] R. Bevans. (2022) Hypothesis testing | a step-by-step guide with easy examples. [Online]. Available: https://www.scribbr.com/statistics/hypothesis-testing/

[31] ——. (2022) Linear regression in r | an easy step-by-step guide. [Online]. Available: https://www.scribbr.com/statistics/linear-regression-in-r/

[32] P. Bhandari. (2022) Extraneous variables | examples, types and controls. [Online]. Available: https://www.scribbr.com/methodology/extraneous-variables/

[33] (2022) Vmware esxi. [Online]. Available: https://www.vmware.com/products/esxi-and-esx.html

[34] F. Middleton. (2022) Reliability vs. validity in research | difference, types and examples. [Online]. Available: https://www.scribbr.com/methodology/reliability-vs-validity/

[35] (2020) Gossipsub v1.1 protocol design, implementation security audit report. [Online]. Available: https://gateway.ipfs.io/ipfs/QmWR376YyuyLewZDzaTHXGZr7quL5LB13HRFnNdSJ3CyXu/LeastAuthority-Gossipsubv1.1FinalAuditReport(v2).pdf

[36] (2022) The go implementation of the libp2p networking stack. [Online]. Available: https://github.com/libp2p/go-libp2p

[37] P. Bhandari. (2022) Sampling bias and how to avoid it | types and examples. [Online]. Available: https://www.scribbr.com/research-bias/sampling-bias/

[38] (2022) gossipsub: An extensible baseline pubsub protocol. [Online]. Available: https://github.com/libp2p/specs/tree/master/pubsub/gossipsub

[39] (2022) Encrochat. [Online]. Available: https://en.wikipedia.org/wiki/EncroChat

[40] (2021) What is testground? [Online]. Available: https://docs.testground.ai/table-of-contents/readme

[41] T. V. Epps, "Testing gossipsub with genesis," 2020. [Online]. Available: https://medium.com/whiteblock/testing-gossipsub-with-genesis-6f89e845b7c1

[42] (2022) go-libp2p-pubsub-tracer. [Online]. Available: https://github.com/libp2p/go-libp2p-pubsub-tracer

[43] M. Mota. (2022) Ethereum development with go. [Online]. Available: https://goethereumbook.org/ethereum-development-with-go.pdf

[44] (2020) Remix ide. [Online]. Available: https://remix.ethereum.org/

[45] T. Perrin. (2018) The noise protocol framework. [Online]. Available: https://noiseprotocol.org/noise.html

[46] Y. Napora. (2022) noise-libp2p - secure channel handshake. [Online]. Available: https://github.com/libp2p/specs/blob/master/noise/README.md

[47] (2022) go-libp2p-pubsub. [Online]. Available: https://github.com/libp2p/go-libp2p-pubsub

[48] M. Seemann. (2020) go-libp2p-pubsub chat example. [Online]. Available: https://github.com/libp2p/go-libp2p/tree/master/examples/pubsub/chat

[49] (2022) Chapter 11 testing regression assumptions. [Online]. Available: https://bookdown.org/jimr1603/Intermediate_R_-_R_for_Survey_Analysis/testing-regression-assumptions.html

[50] (2022) Ubuntu 20.04.5 lts (focal fossa). [Online]. Available: https://releases.ubuntu.com/focal/

[51] (2022) How to turn your raspberry pi 4 into a node just by flashing the microsd card. [Online]. Available: https://ethereum.org/en/developers/tutorials/run-node-raspberry-pi/

A Appendix 1

The ground hardware used for executing the test will be a Dell server which has two Intel Xeon Bronze 3206R Processors with 16 cores each and 128 GB of RAM with a storage capacity of 4 TB. The operating system used to create the virtual machines is a EsXi bare metal host which is a hardware based virtualization platform that then can contain the four machines which will execute the proof-of-concept application, Ethereum full node and collect metrics [33].

Where the simulation environment will be based on Testground [40]. Using an Ubuntu server 20.04 image with 12 CPU cores, 96 GB ram and 500 GB storage for its execution environment [50].

The Ethereum full node will be on a different virtual machine to mimic a real environment where it will use an Ubuntu server 20.04 raspberry PI image which has 8 GB ram, 100 GB storage and 4 Cores [23][51]. The low storage here is just for a test environment and for a real environment it would have to be calculated depending on the storage needs of the Ethereum blockchain.

The metrics collector will be on a third virtual machine where it will use an Ubuntu server 20.04 raspberry PI image which use 4 cores, 26 GB ram and 400 GB storage [42].

B Appendix 1

Github addresses:
https://github.com/luxfeerre/thesis-chatapp

Follow the instructions on the main page for how to use the repository.