UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Security And Cloud Computing

Heikki Santeri Sipilä

Scalability Assessment in Blockchain-enabled IoT Applications

Master's Thesis (30 ECTS)

Supervisors: Mubashar Iqbal, PhD
Abasi-amefon Obot Affia, Msc
Russell W. F. Lai, PhD

Tartu 2023

Scalability Assessment in Blockchain-enabled IoT Applications

Abstract: The use of blockchain in the Internet of Things (IoT) has pawed the way for creating decentralized IoT applications. However, integrating blockchain with the IoT is not straightforward and proposes various challenges. We performed a Systematic Literature Review (SLR) to explore these potential challenges. The SLR resulted in two conclusions. The first conclusion was that potential scalability challenges with blockchain-enabled IoT applications include storage, networking, processing, and throughput challenges. The second conclusion was that the current literature lacks tools and methods to assess these scalability challenges. We address this research gap by first providing a method for creating a scalability assessment tool for blockchain-enabled IoT applications. This method is then showcased in this thesis by designing, implementing, and validating a tool that can be used for scalability assessments of these applications. Our proposed tool uses existing virtualization software to generate simulated IoT devices, which allows for testing the scalability of these applications without the need to use physical devices. We validated the tool by scalability testing a proof-of-concept blockchain-enabled Internet of Vehicles application developed as a part of this thesis. The results of these tests were then analyzed, and the feasibility of the tool was evaluated to be suitable for testing the scalability with a maximum of 24 devices.

Keywords: Blockchain, Internet of Things (IoT), Blockchain-enabled IoT, Scalability

CERCS: T120, Systems engineering, computer technology

Skaleeritavuse hindamine plokiahelaga IoT-rakendustes

Lühikokkuvõte: Plokiahela kasutamine Asjade Internetis (IoT) on loonud tee detsentraliseeritud IoT-rakenduste loomisele. Plokiahela integreerimine IoT-ga ei ole aga lihtne ja pakub erinevaid väljakutseid. Tegime nende võimalike väljakutsete uurimiseks läbi süstemaatilise kirjanduse ülevaate (SLR). SLR tulemuseks oli kaks järeldust. Esimene järeldus oli, et plokiahela toega IoT-rakenduste võimalikud mastaapsuse probleemid hõlmavad salvestus-, võrgu-, töötlemis- ja läbilaskevõimega seotud väljakutseid. Teine järeldus oli, et praeguses kirjanduses puuduvad vahendid ja meetodid nende mastaapsuse väljakutsete hindamiseks. Me tegeleme selle uurimislüngaga, pakkudes esmalt meetodi mastaapsuse hindamise tööriista loomiseks plokiahela toega IoT-rakenduste jaoks. Seejärel tutvustatakse seda meetodit selles lõputöös, kavandades, juurutades ja valideerides tööriista, mida saab kasutada nende rakenduste skaleeritavuse hindamiseks. Meie pakutud tööriist kasutab olemasolevat virtualiseerimistarkvara, et luua simuleeritud IoT-seadmeid, mis võimaldab testida nende rakenduste mastaapsust ilma füüsilisi seadmeid kasutamata. Valideerisime tööriista skaleeritavuse testimisega selle lõputöö osana välja töötatud kontseptsiooni tõestusega plokiahela toega IoV-rakendusega. Seejärel analüüsiti nende testide tulemusi ja hinnati tööriista teostatavust sobivaks skaleeritavuse testimiseks maksimaalselt 24 seadmega.

Võtmesõnad:
Plokiahel, asjade internet (IoT), plokiahela toega asjade internet, skaleeritavus

CERCS: T120, Systems engineering, computer technology

Preface

I want to thank my supervisors for all the help that they have provided. Additionally, I want to thank all the people that have helped me with my life while working on this thesis. Nokia has also offered paid time to work on this thesis, so I want to thank Nokia as well.

Heikki Santeri Sipilä

With the support of the
Erasmus+ Programme
of the European Union

Contents

1 Introduction

Blockchain connects data blocks by appending new transactions on top of previous transactions[1]. This creates a “chain” of immutable blocks. Blockchain accomplishes immutability by using a consensus method where nodes connected to the blockchain network vote on the validity of new transactions [1]. Blockchains saw their first use in cryptocurrencies, such as Bitcoin [2].

IoT applications have been observed to expand to many new areas. For example, IoT devices in in-vehicle applications have been researched [3]. The integration of IoT devices in vehicle infrastructure has allowed large amounts of devices to be connected to the internet and provides different types of data to be processed. However, with the increased amounts of devices and data, the security of systems may become an issue since IoT devices can be vulnerable [4]. Here comes the motivation to use blockchain since it provides various security controls by design, such as decentralized and distributed infrastructure, consensus mechanism, cryptography, and immutable ledger are a few in the list (discussed in detail in Chapter 3).

However, integrating blockchain into IoT applications can introduce scalability challenges because IoT devices have hardware limitations [5, 6, 4]. Following this, blockchain-enabled IoT applications have been noted to have varying scalability issues, such as storage issues [7, 4, 8], consensus issues [5, 6, 9] and processing limitations [10, 8, 11]. Therefore, these scalability issues can lead to increased power usage, waiting times, expensive implementation costs, and distrust in the deployed application.

Additionally, blockchain-enabled IoT applications face the same testing problems as many cloud computing applications, such as creating a physical testing network to evaluate the scalability. A physical testing network can be expensive to implement and difficult to share between many developers if they need to test application changes simultaneously. One previous work utilized virtualization to combat this issue and to test their blockchain-enabled IoT application, PlatiBart [12]. However, the tool presented in their paper was not used to evaluate the scalability of the tested application. Additionally, their tool had some flaws, which are further discussed in this thesis.

1.1 Scope

Blockchain-enabled IoT Applications: The thesis focuses on the scalability assessment of IoT applications that leverage the advantages of blockchain to augment the capabilities of IoT devices, sensors, or systems connected to the internet and sharing data. Thus, his thesis will not assess applications solely utilizing IoT or blockchain.

Scalability Assessment: The definition and characteristics of the term “scalability” is often specific to the domain or system, referring to its adaptability to changes or increases in factors that affect its operation [13, 14]. Therefore, in this thesis, scalability is interpreted as the ability of a blockchain-enabled IoT application to maintain or improve its functionality when exposed to changes or increases in factors that affect its operation.

Scalability-related Parameters: Scalability is often used interchangeably with performance [14]. However, in this thesis, the primary emphasis will be on scalability. Other topics, such as privacy, security, or legal/social issues, will not be the central focus of the study but will be considered mainly for their influence on scalability.

Testing Tool Development and Validation: The thesis will also explore designing a tool for testing the scalability of blockchain-enabled IoT applications. This will include an examination of the technical requirements for the tool, the challenges faced during its development, and methods for validating its effectiveness.

1.2 Problem Statement

In the literature review of this thesis, it was discovered that multiple scalability challenges regarding the scalability of blockchain-enabled IoT applications exist, such as the limited storage and processing power of IoT devices[5, 6, 4], throughput limitations [6, 15, 11] and the inefficiency of the PoW consensus method [16, 6, 17]. In the literature review, we also discovered that there is a lack of studies that assess the scalability of blockchain-enabled IoT applications. Additionally, the current literature lacks tools and methods to assess these applications' scalability. Previous works showed that tests on blockchain-enabled IoT applications were conducted mostly by physical testbeds [18, 19, 20].

The lack of existing methods creates a need for a method for assessing the scalability of blockchain-enabled IoT applications. Additionally, the lack of using virtualization to accomplish scalability assessments creates an opportunity for us to research the use of virtualization in the scalability assessment.

1.3 Research Questions

The main research question for this thesis is the following question. *How to conduct scalability assessment for blockchain-enabled IoT applications?* This research question is divided into the following sub-research questions:

RQ$_1$. *What are the primary challenges and limitations in testing the scalability of blockchain-enabled IoT applications?*

RQ$_2$. *How to develop a scalability testing tool for blockchain-enabled IoT applications?*

RQ$_3$. *How to evaluate the feasibility of the scalability testing tool?*

RQ$_1$ helps to scope out the issues that blockchain-enabled IoT applications currently face. These issues are necessary to understand before moving to **RQ**$_2$. **RQ**$_2$

aims to research how a tool can be implemented that helps research the scalability of blockchain-enabled IoT applications. **RQ**$_2$ is used to understand the problems that the tool development generates. **RQ**$_3$ aims to explore the ways that a created scalability testing tool can be validated for the use case. This research question aims to find possible problems that the created tool faces.

1.4 Contributions

In this thesis, a general-purpose tool is designed, implemented, and evaluated for assessing the scalability of blockchain-enabled IoT applications. The tool allows users to generate a virtual network of simulated IoT devices. It allows the user to configure the device properties, device amount, and the virtual network structure.

This thesis has three main key contributions. Firstly, this thesis provides a design and implementation of a scalability testing tool that can be used to create virtual testing environments to evaluate the performance and scalability of blockchain-enabled IoT applications. This contribution addresses **RQ**$_2$. The second key contribution is the tests conducted on the implemented tool. The first test conducted is a functional test that validates the use of the tool. The second test is a larger-scale test conducted to assess a simulated application's scalability. This contribution addresses **RQ**$_3$. The final key contribution of this thesis is the evaluation of using the tool in real scenarios. This evaluation is done by analyzing the results of both tests conducted using the tool. This contribution further addresses **RQ**$_3$.

1.5 Practical Significance

Our tool was observed to have some practical significance. Previous works which have tested blockchain-enabled IoT devices have mostly utilized physical IoT devices to test their applications[18, 19, 20]. This can be expensive to implement, especially with testing scalability, since the number of devices in scalability tests is greater than in functional tests. Additionally, configuring the physical devices for application testing takes time since the application must be manually deployed to each device and may require further configuration. Our tool allows the automation of IP addresses and application deployment, which speeds up the setup of the testing environment.

1.6 Thesis Structure

Chapter 2 explains the background of the thesis and the required key concepts used throughout the thesis. Chapter 3 reviews existing research and studies concerning scalability issues in blockchain-enabled IoT applications and addresses the first research question, **RQ**$_1$. It presents a synthesized summary of the current understanding in the field, identifying gaps and areas where this research can contribute and establishing the theoretical foundation for the thesis. Chapter 4 focuses on addressing the second research question, $\mathbf{RQ}_2$. The Chapter details developing a scalability testing tool tailored for blockchain-enabled IoT applications. The Chapter includes a detailed explanation of how scalability assessment is approached through the tool and outlines design principles. Chapter 5 continues addressing the second research question, $\mathbf{RQ}_2$, by describing the process of developing the scalability assessment tool, its features, and its application in a real-world context. It also describes the potential challenges faced during implementation. Appendix B provides a user guide for the developed tool. Chapter 6 addresses the third research question, $\mathbf{RQ}_3$, by describing the process and results of validating the scalability assessment framework developed in Chapter 5. It discusses the methods, strategies, and metrics used to determine the effectiveness and efficiency of the developed tool. It outlines the methodology of validation, the results, and their interpretation, presenting the tool's effectiveness and potential limitations. The results from the previous Chapter are discussed in a wider context in Chapter 7, providing an analysis of the findings from the research questions. It provides insights into the scalability issues, the utility of the developed tool, and the validation process results. It also includes a comparison with findings from the literature review within its interpretation of the results and a discussion of the potential implications of the research. Finally, Chapter 8 summarises the research findings and recommendations for future research.

2 Background

This Chapter provides comprehensive background information on the key components of the thesis and explains the main technological tools mentioned throughout the thesis. We begin with the basics of IoT discussed in Section 2.1, and blockchain discussed in Section 2.2, establishing their role in the study. We delve into scalability in blockchain-enabled IoT applications in Section 2.3 and technologies for scalability testing in Section 2.4. Lastly, the Chapter identifies research gaps and outlines the objectives of this thesis in Section 2.5, setting the course for the following chapters.

2.1 Internet of Things (IoT)

IoT has many definitions across research on the subject [21]. However, the consensus about IoT seems to be similar between all the definitions; IoT consists of IoT devices connected to the internet. These IoT devices are physical devices or "things" that can share data [21]. Additionally, IoT devices may be able to process data based on their underlying implementation. These devices often consume small amounts of power since they have limited computational power, connection speeds, memory, and storage. IoT is used to form end-to-end systems, which often share common architectural components with each other[22]. Taivalsaari and Mikkonen [23] proposed the generic model of IoT end-to-end systems. The generic model has four main components: Devices, Gateways, The Cloud, and Applications. Devices are used to collect and share data and to perform actions in the real world, such as actuate switches. The gateways then connect the cloud to these devices. The cloud is then used to store, manage, and process data. Finally, the cloud provides application data to the final component, the application, which then presents this data to the user as a web or a mobile application.

These systems can be used for multiple applications, such as smart homes and healthcare. In recent years, the integration of blockchain technology with IoT has allowed IoT applications to expand into diverse areas, showcasing its potential in various domains, such as the Internet of Vehicles (IoV) [3], drones [24], industrial control systems [25], and smart grids [26].

One recent prominent use case for IoT is the Internet of Vehicles (IoV), which will be utilized in the validation Chapter of this thesis. IoV originated from Vehicle Adhoc Networks (VANETs), which are communication networks where each vehicle acts as a router and thus creates a mobile network [27]. However, these networks were deemed as unstable and random since the connections between vehicles are challenging to maintain [3]. To solve this challenge, IoV networks are formed from a combination of mobile network connections (such as LTE), VANETs, and vehicle intelligence[3]. The vehicle intelligence of IoV networks aims to intelligently integrate humans and vehicles by utilizing network technologies, such as machine learning, artificial intelligence, and swarm computing [3].

2.2 Blockchain

We use the term blockchain widely in this thesis. This section explains what blockchain is and showcases the Ethereum protocol [28]. After, we discuss the concept of a blockchain-enabled IoT application.

Blockchain consists of chained blocks of data containing previous transactions' data. New transactions append data on top of the chain, thus creating an append-only data structure [1]. These transactions contain a hash, timestamp, and nonce to validate each block on the chain. The nodes in a blockchain check the validity of new transactions by using a consensus mechanism. Once a transaction is sent, the transaction is checked by the peers in the network by using the consensus mechanism. If most nodes agree that the transaction is valid, then the transaction is appended to the block, which ensures immutability [1].

2.2.1 Ethereum Protocol

One example of a blockchain protocol is the Ethereum protocol. The Ethereum protocol is designed to be used in decentralized applications [28]. The state of Ethereum chains is made from transactions sent by Ethereum accounts. There have two types of accounts with Ethereum: externally owned accounts controlled by private keys and contract accounts controlled by contract code. These accounts can then send transactions to the network, where the transaction contains the following data fields:

- Public address of the recipient
- Signature of the sender
- Ether transfer amount
- STARTGAS value, which represents the maximum amount of computational steps that the transaction execution can take
- GASPRICE value, which represents the value per computational step required for the transaction
- External data (optional)

Transactions are then validated by the peers that are connected to the network. For this validation, they receive a small fee deducted from the transaction. The Ethereum protocol is used in this thesis to validate the implemented tool. The usage of Ethereum is further discussed in Chapter 6.

2.2.2 Blockchain-enabled IoT Applications

In this thesis, the concept of blockchain-enabled IoT applications is used to describe an application that runs on IoT devices and utilizes blockchain in the application. These applications share some aspects of the previously mentioned common end-to-end architectural model [23]. However, blockchain-enabled IoT applications use blockchain to replace or in combination with the Cloud component of the previously mentioned generic architectural model [23].

2.3 Scalability in Blockchain-enabled IoT Applications

Scalability is highly dependent on the application domain and should be explored in the context of high-level business goals [13, 29]. Scalability, in the context of blockchain-enabled IoT applications, refers to the system's ability to handle an increase in workload or demand without compromising expected functionality or operations. It involves understanding how changes in various aspects of the system's environment and design can impact its overall quality as these aspects vary over expected operational ranges [13, 14].

Learning from [14], primary factors affecting the scalability of blockchain-enabled IoT applications reflect characteristics of the application domain and the machine that will affect the system behavior. These factors include scaling factors, non-scaling factors, and nuisance variables. Scaling factors are the characteristics of the application domain or the system that can be changed to affect the system's behavior significantly [14]. In the case of blockchain-enabled IoT applications, the number of devices, the volume of transactions, and the complexity of the transactions [30]. Non-scaling factors are characteristics of the application domain or the system that are either set to fixed levels or varied in a nominal scale to enable the system to deal with the scaling dimensions [14]. These should support the scaling dimensions. These include the consensus mechanism used, the type of IoT devices used, or the operating system used [31]. Nuisance variables impact system behavior but aren't controllable within a scalability experiment if the infrastructure is predetermined and unalterable [14]. These include processor speed or other machine configurations of this nature.

These factors can be manipulated over operational ranges for the scalability analysis as independent variables causing system behavior that can be measured to derive dependent variables [14]. The dependent variables represent aspects of the system's behavior that can be measured to show how the system is affected by changes in the independent variables [14]. They typically correspond to software metrics related to performance, cost, maintenance, reliability, security, and operational constraints. For blockchain-enabled IoT applications, the dependent variables might include latency and throughput. One can plot these variables to show the scalability relationship [14].

Scalability takes on a complex dimension, requiring consideration of factors specific to blockchain and IoT. Blockchain scalability emphasizes transaction throughput, network latency, and data consistency [32]. In contrast, IoT scalability focuses on managing numerous connected devices, real-time data processing, and the overall reliability of the system [33]. In the combined environment of blockchain-enabled IoT applications, scalability testing should address both these sets of concerns ensuring that the system can handle an increasing number of scaling parameters without any degradation in its operation. We discuss scalability testing considerations in Section 2.4.

2.4 Scalability Testing in Blockchain-enabled IoT

Scalability testing in blockchain-enabled IoT represents a multifaceted challenge that requires an integrated approach [34]. By recognizing and addressing the distinct and interconnected aspects of blockchain [35] and IoT scalability [36], practitioners can develop robust and scalable systems that leverage both technologies' strengths. The convergence of testing practices offers comprehensive means to tackle complex scalability challenges inherent in this exciting intersection of blockchain and IoT.

Testing in blockchain requires simulating various network conditions, transaction loads, and consensus algorithms [32]. On the other hand, IoT testing necessitates emulating different device types, communication protocols, and data volumes [33]. Both physical and virtual testbeds might be employed when considering the combined environment. These testbeds must focus on simulating complex real-world scenarios encompassing diverse devices and fluctuating transaction loads, providing flexibility, cost-efficiency, and overcoming physical limitations [12]. The choice between physical and virtual testbeds for scalability testing can have significant implications. For physical testbeds, testing on physical hardware like Raspberry Pi and servers provides a real-world understanding of how the system will perform. However, it can be costly, less flexible, and limited by the hardware itself [20]. On the other hand, virtual environments offer flexibility, cost-efficiency, repeatability, and the ability to simulate complex scenarios. Virtual testbeds allow for more nuanced scalability testing without physical constraints, making them a preferable option for many researchers [12]. Thus, moving towards virtual testbeds provides a path for more flexible, cost-effective, and comprehensive testing solutions and will focus on such testing strategies [37]. To build a suitable virtual testing platform, we discuss the decentralized interaction facilitated by the Geth client, the virtualized environments offered by VirtualBox, and the automation provided by Vagrant, each contributing to creating a scalable, flexible, and cost-effective testing landscape. These tools allow for intricate simulations and promote repeatability and extensibility, factors essential for rigorous and accurate scalability testing. Including these elements within the thesis underscores the intricate network of technologies that combine to explore and assess the scalability of blockchain-enabled IoT applications, marking a significant contribution to the field.

2.4.1 Geth Client

Since Ethereum is a protocol, it does not have a centralized client that is used to interact with the chain. Instead, many open-source clients exist that can be used to interact with the Ethereum chain. One example of these clients is the Geth client [38], which is written with the Go programming language. The Geth client is an Ethereum client that allows the user to create Ethereum accounts, create Ethereum transactions, mine the Ethereum chain, configure the network to which the client is connected, and query the chain. The Geth client is used in this thesis as a part of the validation tests of the implemented tool. Further explanations of using the Geth client are presented in Chapter 6.

2.4.2 Virtualbox and Vagrant

We implement and validate the built tool utilizing VirtualBox to generate virtual IoT devices. VirtualBox is an open-source virtualization product that allows the virtualization of multiple different systems [39]. VirtualBox allows to run multiple Virtual Machines (VMs) parallelly at the same time. Additionally, VirtualBox can simulate network connectivity, processors, storage, and memory on a Virtual Machine. VirtualBox's usage and limitations are further discussed in Chapter 5.

Configuring VMs manually in VirtualBox is time-consuming and does not have built-in automation tools. To combat this, Vagrant provides a tool that can be used to automate the building and management of virtual machines in VirtualBox[40]. Vagrant provides virtual machines with a `"Vagrantfile"`. Within this file, many aspects of the virtual machines can be modified. The first aspect that can be specified is the virtual machine's OS. Vagrant can download an operating system (OS) hosted in the cloud or utilize a locally built OS. Vagrant can also be used to build a custom image of an OS, which can be later used in other instances. Vagrant also allows the user to choose a virtual machine provider, which can differ from the used Virtualbox. Officially, vagrant supports VirtualBox and VMware as VM providers. In this thesis, Vagrant is used to automate the generation of virtual IoT devices in VirtualBox. The use of Vagrant in this thesis is further described in Chapter 5.

2.5 Research Gaps and Objectives

While research has been into blockchain-enabled IoT applications, the scalability challenges remain under-explored [41, 42]. Studies have highlighted the potential benefits of using blockchain in IoT scenarios. Still, few explore comprehensive scalability testing, which forms a crucial gap in the existing research. Moreover, there are limited testing environments [12] for assessing the scalability of blockchain-enabled IoT applications. Thus, this can hinder a thorough understanding and testing of scalability within this scope and has restricted the advancement of efficient solutions. Following this, in this work, we aim to deepen the understanding of scalability issues in blockchain-enabled IoT applications and contribute to the existing body of knowledge by the following objectives:

- Conducting a systematic literature review (SLR) of the current state of research on the scalability of blockchain-enabled IoT applications, contributing to answer $\mathbf{RQ}_1$. This review will explore the various scalability issues that have been identified, as well as the strategies that have been proposed to address them. This review also provides information on existing testing environments designed for blockchain-enabled IoT applications, contributing to answer $\mathbf{RQ}_2$. This investigation will help identify the strengths and weaknesses of existing approaches, providing essential insights for developing a more effective scalability testing tool.

- Designing and implementing a scalability testing tool tailored for blockchain-enabled IoT applications, contributing to answer $\mathbf{RQ}_2$. The tool's design will consider the insights gained from the literature review and examine existing testing environments.

- Validating the performance of the proposed tool. This involves implementing tests within IoV use cases to confirm its efficacy in testing the scalability of blockchain-enabled IoT applications, answering $\mathbf{RQ}_3$.

3 Literature Review

This Chapter delves into the SLR conducted as part of this thesis. The methodological approach used for the review is detailed in section 3.1. Subsequently, the findings from the review are discussed in section 3.2. The results are discussed in Section 3.3, followed by a summary in Section 3.4. Although previous research has addressed blockchain-enabled IoT applications [41, 42], two significant gaps persist. Firstly, the focus on scalability issues within blockchain-enabled IoT applications is notably scant in current literature. Secondly, there is a paucity of tests exploring the scalability of such applications. This literature review investigates scalability issues inherent in blockchain-enabled IoT applications and their suggested solutions and examines existing testing environments proposed for these applications. Testing is a crucial phase in deploying scalable blockchain-enabled IoT applications. Traditionally, this would involve physically setting up a network of IoT devices. Nevertheless, advancements in modern hardware allow for the possibility of simulating the entire environment. Hence, the literature review also examines contemporary methods for testing blockchain-enabled IoT applications. Given the thesis's relevance to software engineering, the systematic literature review will adhere to the guidelines proposed by the University of Keele for software engineering [43]. These guidelines provide a well-structured approach to conducting a systematic and thorough review of the existing literature in the field.

3.1 Review Method

The literature review aims to comprehensively understand scalability issues in blockchain-enabled IoT applications, their proposed solutions, and existing testing methods and frameworks. This knowledge lays the foundation for designing and implementing a virtual testing tool for blockchain-enabled IoT applications.

3.1.1 Data Sources

Our main sources of information for this literature review are IEEE Xplore, ACM, and Scopus databases. These databases were chosen for this SLR because they are widely recognized for their comprehensive, high-quality content in technology and engineering fields, including areas relevant to this research, such as blockchain and IoT.

3.1.2 Search Strategy

Finding relevant literature for $\mathbf{RQ}_1$ differs from finding literature for $\mathbf{RQ}_2$. It requires separate keyword searches due to its distinct nature. Consequently, two separate literature searches were conducted.

For **RQ**$_1$, we needed to identify studies that focus on scalability issues of blockchain-enabled IoT applications. Each database allows for an advanced search, enabling us to include or exclude certain keywords from specific parts of the literature. To pinpoint studies addressing scalability challenges in IoT-based blockchain applications, it was necessary to search for papers on general challenges in this area and filter out those that do not address scalability. Targeting these studies is complex, as IoT and blockchain are popular keywords in recent literature. For this search, we focused only on titles, allowing us to effectively exclude studies not directly related to the challenges of blockchain-enabled IoT applications. The keywords included different forms of "IoT" (such as “Internet Of Things” and “Internet-Of-Things”), “Blockchain”, and synonyms for challenges (e.g., “Challenge”, “Problem”, “Issue”). Due to the vast amount of literature in this area, we limited our selection to papers cited by ten or more studies, ensuring that the identified issues are acknowledged in the field.

RQ$_2$ focuses on the testing of blockchain-enabled IoT applications. Although it shares some keywords with the first question, this search specifically targets literature related to testing. In addition to the first two keyword groups, we added “test” (including variations like “testing”) to our search. The detailed research strategy is illustrated in Figure 2.

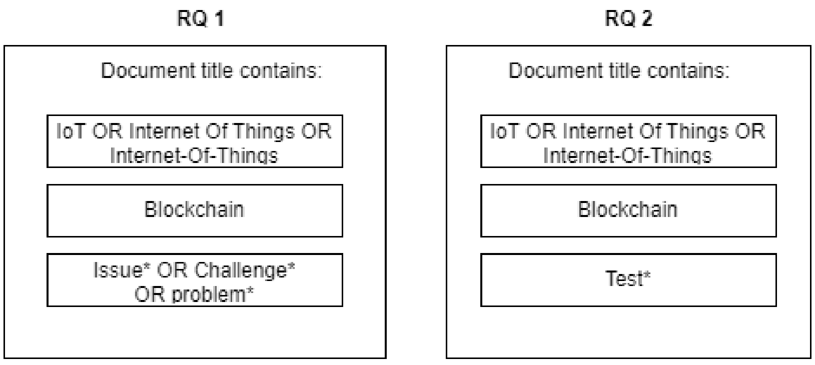
RQ 1
Document title contains:
IoT OR Internet Of Things OR Internet-Of-Things
Blockchain
Issue* OR Challenge* OR problem*
RQ 2
Document title contains:
IoT OR Internet Of Things OR Internet-Of-Things
Blockchain
Test*

Figure 2. Summary of the research strategies for each research question

This strategy is then applied to each database. The resulting search query for each database and the number of results they produce are shown in Table 1 for **RQ**$_1$ and in Table 2 for **RQ**$_2$.

3.1.3 Selection Criteria

The results of the SLR may contain duplicates since databases may have shared entries. To combat this, all results were checked for duplicate studies. After filtering duplicate results, a total of 28 studies for **RQ**$_1$ and six studies for **RQ**$_2$ were left. The search results

Database	**Query method**	**Results (cited $\geq$ 10 times)**
IEEE Xplore	("Document Title": IoT OR "Document Title":Internet Of Things OR "Document Title":Internet-Of-Things AND ("Document Title":Blockchain) AND ("Document Title":Issue*, "Document Title":Challenge* OR "Document Title":Problem*)	7
ACM	[[Title: iot] OR [Title:internet of things] OR [Title:internet-of-things]] AND [Title:blockchain AND [[Title: issue*] OR [Title:challenge*] OR [Title:problem*]]	8
Scopus	(TITLE: (IoT OR internet AND of AND things OR internet-of-things) AND blockchain AND (issue* OR challenge* OR problem*))	21

Table 1. Query methods for $\mathbf{RQ}_1$ and count of results

Database	**Query method**	**Results**
IEEE Xplore	("Document Title": IoT OR "Internet Of Things" OR "Internet-Of-Things") AND ("Document Title": Blockchain) AND ("Document Title": Test*)	2
ACM	[[Title: iot] OR [Title: internet of things] OR [Title: internet-of-things]] AND [Title: blockchain] AND [Title: test*]	2
Scopus	(TITLE (IoT OR internet AND of AND things OR internet-of-things) AND TITLE (blockchain) AND TITLE (test*))	2

Table 2. Query methods for $\mathbf{RQ}_2$ and count of results

can contain studies that do not have research regarding the research questions. To combat this, the results are further refined by filtering the studies using the following exclusion criteria:

- Based on the title and abstract, a study will be excluded if the study does not discuss research on issues/solutions to IoT-based blockchain
- Based on the title and abstract, a study will be excluded if the study does not discuss research on testbeds for blockchain-enabled IoT applications.

This filtering resulted in the exclusion of two studies. The studies [44, 45] were excluded since both of them discuss the issues of IoT but do not discuss blockchain-enabled IoT applications. The rest of the studies were included since they did not meet the exclusion criteria. This resulted in a total of 32 studies for both research questions.

The goal of this literature review is to have an overview of the scalability issues that blockchain-enabled IoT applications have and to find out previous research on blockchain-enabled IoT application testbeds. Thus the following data aspects are extracted from the included studies:

- Issues in blockchain-enabled IoT applications
- Possible solutions to these issues
- Architecture, virtualization software, and OS on nodes used in a testing environment.

3.2 Review Results

The literature review for **RQ**$_1$ shows that issues with blockchain-enabled IoT applications have a wide range of existing works resulting in 26 studies after review execution and extensive application of our selection criteria. The literature review for **RQ**$_2$ resulted in only six studies. The included databases do not include much literature regarding testing blockchain-enabled IoT applications. This can be a result of 2 different reasons. Firstly, the search keywords may be too strict and thus filter out studies regarding IoT-based blockchain testing. The second possible reason is that IoT-based blockchain testing research is not widely researched. This section presents the results of this review contributing to our research objectives.

3.2.1 Results for RQ$_1$: Scalability

Our review for **RQ**$_1$ focused on scalability issues of blockchain-enabled IoT applications and solutions to these issues. The literature examined scalability issues emerging as a key concern when addressing the growth and operation of blockchain-enabled IoT applications. While scalability issues are the main focus, security and privacy issues cannot be disregarded as they become significant when designing and implementing a blockchain-enabled IoT application for a growing user base. As the scale of the system expands, the complexity of maintaining security and privacy grows, making these issues crucial to address. In addition, social and legal issues also emerge as the system scales up. Although these are not intrinsically linked to scalability concerns, they become increasingly important as the user base grows.

Scalability issues and solutions: From our review, we find that scalability issues primarily stem from the limited storage and processing power of IoT devices [5, 6, 4], throughput limitations [6, 15, 11], and PoW inefficiency in blockchain-enabled IoT [16, 6, 17], and the network limitations of IoT devices [46, 47, 4]. We delve into these problems and their proposed solutions, summarizing the issues in Table 3 and their corresponding solutions in Table 4.

Study	Scalability issues				
	Limited storage	Limited processing power	Throughput	PoW inefficiency	Network limitations
[7]	x				
[4]	x	x			x
[8]	x	x		x	
[5]	x	x			
[48]	x	x			
[6]	x		x		
[15]	x		x		
[47]	x				x
[9]	x				
[49]		x		x	
[50]		x			
[16]		x			
[51]		x	x		
[11]			x	x	
[10]				x	x
[17]			x	x	
[52]			x		
[53]			x	x	
[54]				x	
[55]				x	

Table 3. Scalability issues of blockchain-enabled IoT applications

Study	Computation Offloading	Data Offloading	Optimized Consensus Method	Data Compression
[52]	x			
[5]	x	x	x	
[46]		x		
[7]		x		
[11]			x	
[56]			x	
[50]				x

Table 4. Solutions to combat scalability issues of blockchain-enabled IoT applications

A primary scalability issue in blockchain-enabled IoT applications is the limited storage capacity of IoT devices [5, 6, 4]. As more devices interact with the application, the blockchain's size increases, making it difficult for devices with limited storage to store the entire blockchain. A solution involves offloading data to machines with greater storage capacity [5, 46, 7]. This means that application data is not stored on the chain itself but may be hosted by a cloud. The blockchain itself is then used to verify data transactions. Another way of implementing data offloading would be to shard data.

Another scalability issue is the limited processing power of IoT devices. Cryptographic operations in IoT devices can be computationally intensive, affecting transaction latency and risking transaction overflow if throughput is outpaced by incoming requests [16, 6, 17]. Throughput limitations also pose scalability issues [16, 6, 17]. This is because the limited processing power of IoT devices can also create bottlenecks in transaction processing and overflow of transactions, potentially leading to application failure. Offloading computations to a more powerful server is one solution [52, 5], although this is more suited to applications where latency is not critical due to transmission delay. Alternately, switching to a less computationally intense consensus method like proof-of-stake could reduce latency and improve throughput [46, 5, 11].

The inefficiency of proof-of-work (PoW) consensus in blockchain-enabled IoT applications is another scalability concern. The computational intensity of PoW could be mitigated by implementing proof-of-stake, which focuses only on transaction verification and hence requires fewer computations[46, 5, 11].

Finally, the network limitations of IoT devices present scalability issues [46, 47, 4]. Often connected via wireless interfaces with limited performance, the data transfer volume on the blockchain could bottleneck the application's throughput. This limitation underscores the need for network infrastructure improvements or efficient data handling protocols to ensure scalability. One solution to this issue mentioned in the literature was to use data compression on the blockchain [55]. Data compression could mitigate the effects of limited networking interfaces.

Security and privacy related issues and solutions: Security and privacy are paramount concerns for blockchain-enabled IoT applications. One frequently mentioned issue pertains to the security of individual devices [55, 6, 9]. With limited processing power and memory, these devices often overlook security in favor of functionality. Weaknesses such as susceptibility to buffer overflow attacks and exploits in their operating systems can compromise security. Additionally, the use of default passwords and keys further elevates the risk. When scaling a blockchain-enabled IoT application, these vulnerabilities could allow an attacker to forge transactions by gaining access to a node's private keys or even hijack many devices and seize the entire blockchain. The risk of combined attacks against blockchain-enabled IoT applications is another critical concern [54, 6]. In such a scenario, an attacker could exploit compromised devices to take over the entire chain, potentially disrupting the entire application's reliability and scalability. Using smart contracts in blockchain-enabled IoT applications can introduce additional vulnerabilities [7, 53]. If these contracts are exploited, an attacker could alter the chain's state, impacting application reliability and, thus, scalability.

User privacy is another crucial issue frequently highlighted in the literature [57, 58, 55] [53, 59, 46]. The immutability of blockchain data can potentially compromise privacy if sensitive user data is stored on the chain. Another privacy issue is that the writers of the chain are pseudonymous. If the applications write data to the chain containing personally identifiable information, it could link pseudonymous addresses to individual users and compromise user anonymity. Such privacy concerns could limit application scalability as users hesitate to use an application that compromises sensitive information. Lastly, transaction privacy within blockchain-enabled IoT applications poses another potential problem [60, 53]. Transactions on the chain could inadvertently leak application data, which can then be used to link actions to public wallet addresses. The transparency of blockchain transactions might then lead to privacy breaches, which can also affect scalability.

Social issues and solutions: As a relatively new field, blockchain-enabled IoT applications face several challenges regarding legal and technical standards [48, 59, 49, 4, 11]. The decentralized nature of blockchain makes data ownership determination and deletion problematic, and the dearth of legal norms could lead to applications being shuttered due to non-compliance or developers shying away from blockchain due to potential legal ramifications. Technical standardization is also lacking in blockchain-enabled IoT applications [11, 50, 48]. While not a scalability issue per se, the absence of standards could hinder performance optimization and inflate the costs of implementing blockchain applications. The absence of these norms contributes to high implementation and maintenance costs for blockchain-enabled IoT applications [51, 16, 11]. For instance, in blockchain applications utilizing Ethereum [28], nodes need to pay for each transaction or “gas”. While not directly affecting scalability, these costs could limit application development, delaying research and solutions to blockchain-related issues.

3.2.2 Results for RQ$_2$: Scalability Testing Tools

The literature search for blockchain-enabled IoT application testbeds resulted in six studies. Three studies [18, 19, 20], showcase testing on a physical testing environment, one study [12], showcases a virtual testbed, and others did not present any testbed experiments [61, 62]. Mitra et al.[18] presented a way of using Raspberry Pi combined with light servers to test and run a blockchain-enabled security framework. Their framework transfers wildlife monitoring data from drones equipped with IoT devices to a monitoring station. The architecture of their experiment is presented in Figure 3. The blockchain that was run on the experiment was custom-built. Their experimental test setup used a single Raspberry Pi connected to a single light server to test their framework as a proof of concept.

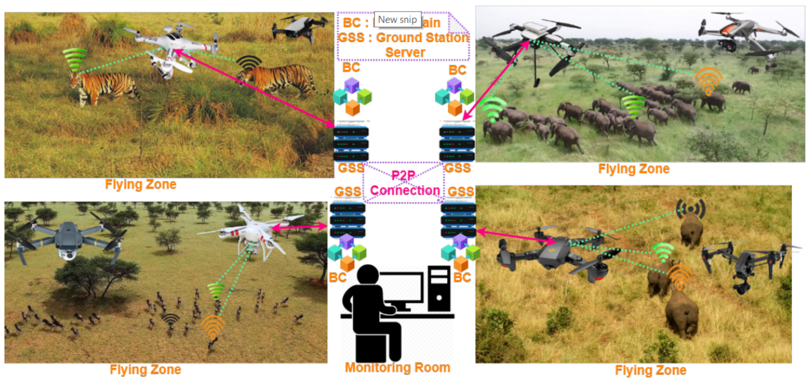

Figure 3. System model of a IoT-enabled application for monitoring wildlife [20]

Wang et al. [19] presented a blockchain-enabled IoT application that was tested on a physical testbed[19]. The tested application was an Ethereum-based IoT application that stores sensor data in the chain. The testbed that was implemented was more complex than the previous experiment. It consisted of 4 light server nodes that were run on two laptops (running MacOS) and two desktop PCs (Running Ubuntu 16.04). In addition to this, each light server was connected to a Raspberry Pi 3, running raspbian OS.

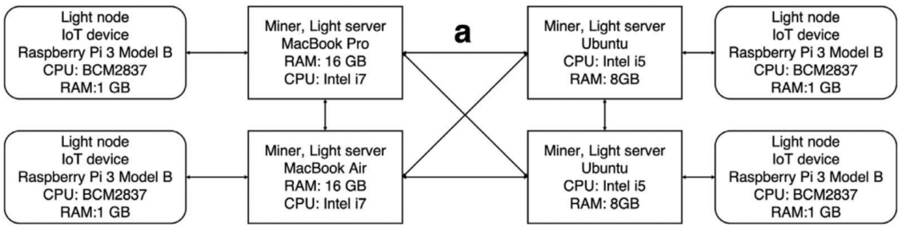

Figure 4. Physical test [20]

Sun et al. [20] implemented an IoT-based blockchain solution for validating Electric Vehicle (EV) battery swapping [20]. Their solution used Ethereum as the blockchain for the system. The system was designed to equip each EV with an IoT device. The service stations that swap batteries would have a desktop PC to handle the confirmations of battery swaps. Tests for this system were completed using a PC and a raspberry pi. Tests were run in 3 different software configurations.

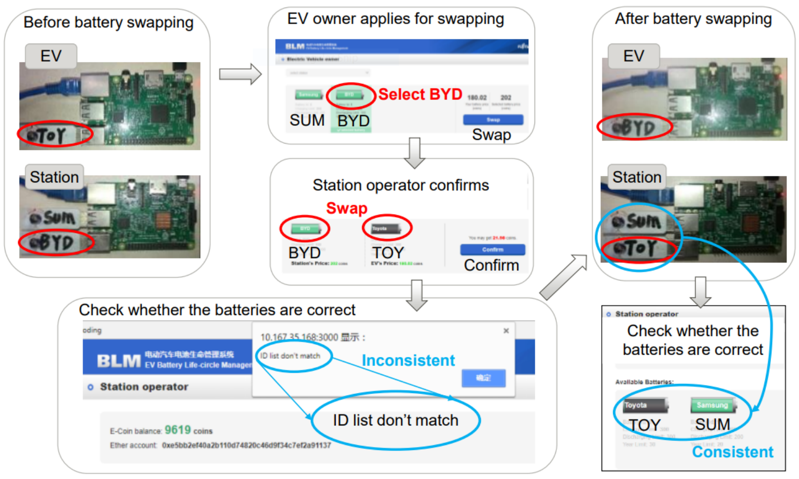

Figure 5. System flowchart of Sun et al. [20]

One study that did design a virtual testbed for blockchain-enabled IoT applications was conducted by Walker et al. [12]. Their proposed platform used a previously created Resilient Information Architecture Platform for Decentralized Smart Systems (RIAPS) [63] combined with a custom domain-specific language to create a Platform for Trans-active IoT Blockchain Applications with Repeatable Testing (PlaTIBART) [12]. The platform was created for regression testing for blockchain-enabled IoT applications. The study also mentioned extending the platform to support other blockchain technologies.

Based on our review, most of the tests conducted in the review were small-scale tests completed with physical hardware that was done to prove the feasibility of those applications [18, 19, 20]. The exception was the PlaTIBART [12], which was used to simulate many IoT clients using Ethereum. This platform also demonstrated that it scales successfully as clients are added. However, the platform had some limitations:

- PlaTIBART only supports Ethereum-based IoT applications. Ethereum is a commonly used solution for many blockchain-enabled IoT applications since it allows the flexible use of smart contracts. However, other blockchain technologies, such as Hyperledger fabric (HLF) [64] may be more suitable for blockchain-enabled IoT applications. This leaves a gap in the literature for creating a testing framework that supports IoT applications that do not use Ethereum.

- IoT clients are simulated without an operating system; instead, they are run as plain Geth clients. This makes it possible to test the application logic. However, it neglects the possibility that an IoT client may not have the processing capabilities to complete that implemented logic. The authors even remark that their solution is a testbed for regression testing. This leaves the opportunity to create a virtual testbed that could be used for scalability testing.

- Network connections are handled as ssh connections between each machine. This allows all of the simulated machines to be connected, but it does not allow the testers to modify network delays between machines. Additionally, since the network connections are encrypted, it would not allow security testing, for example, in cases where an attacker modifies the application packets sent from the IoT node.
- PlaTIBART requires a lot of manual configuring for networking settings due to their custom DSL implementation for network management. This can make testing with many nodes tedious; for example, in cases where 1000 or more devices need to be tested, writing that custom DSL for the network configuration would take a long time. This allows for automating IoT node creation that could be used in a testbed. This would allow for testing with various amounts of nodes.

3.3 Discussion of Results

The literature review's results show sufficient studies regarding the issues of blockchain-enabled IoT applications. These studies made it possible to determine scalability issues, from which the different factors affecting blockchain-enabled IoT applications were analyzed. The literature review proved that some experiments have been conducted regarding testing blockchain-enabled IoT applications. However, there seems to be a gap in the literature assessing the scalability of blockchain-enabled IoT applications. To address this gap, we propose the following 4-step method to assess the scalability of blockchain-enabled IoT applications:

1. **Choosing a scalability property to test.**
   In this step, the scalability property to test is chosen. One way of measuring the scalability of blockchain-enabled IoT applications can be measured from the relation of the number of requests to the application's response time. The limitations in processing power [5, 6, 4], throughput limitations [6, 15, 11], consensus inefficiencies [16, 6, 17], and the network limitations of IoT devices [46, 47, 4] all are affected by the number of requests that are received by the application. Thus, the relationship between the number of concurrent requests and the application's response time can be used to determine scalability.

   Another way scalability can be measured is from the relation of the size of the chain to the application response time. The limited storage and processing power of IoT devices [5, 6, 4], throughput limitations [6, 15, 11], and the network limitations of IoT devices [46, 47, 4] are all affected by the increased size of the chain. Thus, the relationship between the size of the chain and the time for a successful request can be used to assess the application's scalability.

2. **Building a testing environment.**
   In this step, a testing environment should be built where devices running the appli-

cation can be configured. There are two options for building a testing environment. The first option would be to utilize physical devices for the testing environment, as shown by previous experiments in the domain [18, 19, 20]. The second option is to utilize virtualization to simulate these devices, as shown by PlaTIBART [12].

3. **Running the test on the testing environment.**
   In this step, the chosen property should be tested on the environment. For example, if the effect of the number of clients on the application was chosen, then it should be tested how the increase in clients affects the response time of the application. Similarly, if the effect of the increased size of the blockchain on the application was chosen, then the increase should be compared to the application's response time.

4. **Analyzing the results of the test.**
   Once the results of the tests are acquired, the results should be analyzed. The first property that should be analyzed is that are the test results valid. Especially with virtualized testing environments, they may not represent actual application behavior if the virtualized environment is not setup correctly or if the host machine does not have enough processing capability to complete the test. Then once the results have been deemed feasible, the correlation between the number of requests and the application's response time should be analyzed. This correlation provides information about the scalability of the application.

The second step requires the creation of a testing environment. However, creating this environment may be costly if the devices need to be set up physically. One way of solving this would be to simulate the devices as shown by PlaTIBART [12]. However, as discussed before, PlaTIBART had some flaws. Thus, there exists a need for a tool that can be used to create testing varying environments for scalability tests of blockchain-enabled IoT applications. Based on this need, we chose to develop an environment-creation tool that could be used to assess the scalability of blockchain-enabled IoT applications.

3.4 Summary

This Chapter presented the SLR that was conducted for this thesis. The literature review provided answers to the $\mathbf{RQ}_1$ by reviewing current literature. We identified that blockchain-enabled IoT applications suffer from the following four main issues:

1. Limited storage and processing power of IoT devices [5, 6, 4]
2. Throughput limitations [6, 15, 11]
3. PoW inefficiency in blockchain-enabled IoT [16, 6, 17]

4. Network limitations of IoT devices [46, 47, 4]

The SLR also answered **RQ**$_2$ by finding studies where tests for existing blockchain-enabled applications were conducted. Only six conducted tests were found, of which only one utilized virtualization.

We then used these results to propose a framework that can be used to assess the scalability of blockchain-enabled IoT applications. The framework uses a 4-step method to assess the scalability of a blockchain-enabled IoT application. The proposed framework includes a step that requires the creation of a testing environment. Since there is a lack of tools regarding this area, we decided to research the creation of this type of tool.

4 Research Method and Design

This Chapter explains the research method and design process of creating the scalability assessment tool. In the previous chapter, it was specified that there exists a need for an environment creation tool for assessing the scalability of blockchain-enabled IoT applications. Section 4.1 discusses our research method for this tool. Section 4.2 explores the possible stakeholders of the tool and discusses stakeholders' goals that guide tool design. Specification of the tool is presented in Section 4.3, and lastly, Section 4.4 overviews the design requirements of our tool.

4.1 Research Method

This thesis uses a 3-step process for creating a scalability assessment tool for blockchain-enabled IoT applications. The 3-step process is described below:

Step 1: Tool design

- Identification of tool stakeholders and Goal modeling
- Creating specifications for the tool
- Creating requirements for the tool

Step 2: Implementation

- Choosing the technological components
- Implementing the tool

Step 3: Verification and feasibility

- Verification test 1
- Verification test 2

– Analyzing the feasibility

The first step is the tool design. The first part of this step is to identify the possible stakeholders of the tool. Then, based on the possible stakeholders, the goals of these stakeholders are modeled. Based on these goals, design specifications, and requirements are created for the tool's implementation. The second step in the process is to implement the tool. First, the underlying technological components, such as already available virtualization software, are chosen. Then, the implementation of the tool is completed. The third step of creating the tool is to verify that the tool functions and estimate the feasibility of using the tool for testing the scalability of blockchain-enabled IoT applications. The first verification test is used to set up and run a blockchain on the tool to verify that the basic functionality exists in the tool. The second test is used to complete a larger scale test, where the proposed scalability analysis can be conducted on whether the tool accomplishes the stakeholders' goals. Finally, the tool's feasibility is estimated based on the results of the two tests.

4.2 Identifying Possible Stakeholders and Modelling goals

This section presents the possible stakeholders of a blockchain-enabled IoT scalability testing tool. From these stakeholders, the goals of each stakeholder are modeled.

The first possible stakeholder for a blockchain-enabled IoT application testing tool is an application developer. An application developer is a potential stakeholder in a scalability testing tool since application developers may want to assess the scalability of their application. The second possible stakeholder for a blockchain-enabled IoT application scalability testing tool is an application tester. An application tester is a potential stakeholder of the tool since application testers may need to do scalability testing of a developed blockchain-enabled IoT application.

Based on the previously mentioned potential stakeholders, goals for the tool are modeled. These goals are then transformed into specifications and requirements for the proposed tool. Since this thesis aims to implement a tool that helps assess the scalability of blockchain-enabled IoT applications, only aspects revolving around assessing scalability are focused on.

Application developer goals: The application developer has a main goal, which can be divided into sub-goals. The main goal of the application developer is to develop scalable blockchain-enabled IoT applications efficiently. To do this, we identified that the application developer has the following sub-goals; efficiently develop, test, and deploy the application. From these sub-goals, testing the application is the most relevant since the proposed tool is used to assess the scalability of these applications. The sub-goal of testing the application contains many further sub-goals. However, concerning this thesis, the tester's most important aspect is testing the application's scalability. To test the scalability, an application developer should be able to deploy the developed application and configure the number of users using a tool.

Application tester goals: The main goal of an application tester is to test and validate the application. The sub-goals to achieve this are as follows; develop tests for the application, run the tests on the application, and Evaluate the results. From these subgoals, the goals of running the tests and evaluating the results of those tests are the most important goals for the tool. Similar to the application developer goals, these subgoals can be further broken down into many sub-goals. The application tester must have a testing environment to develop tests for the application. To run the tests on the application, the tester should have the application and the environment. To evaluate the tests, the environment and tests should be repeatable and allow for measuring different properties of the applications during the tests.

4.3 Specification of the Tool

Testing the scalability of blockchain-enabled IoT applications is a complex process. Blockchain-enabled IoT applications combine many elements of system and network hardware and software. This can make it difficult to test and troubleshoot these applications. A common way to speed up and save costs while testing interconnected applications is to virtualize these systems using machines or software like Vagrant, Kubernetes, or Docker. A virtual testbed offers some benefits when compared to a physical testbed.Firstly, it can reduce the cost of setting up the environment since the hardware does not have to be purchased. The second benefit of virtual testing environments is that tests are easier to recreate. Since virtual environments can be automated, misconfigurations are less likely when configuring tests. The final reason for virtualizing the test environment is that it can speed up development and fixing the application. The tool's specification is to create a virtual testing environment where the scalability of blockchain-enabled IoT applications can be assessed (Fig. 6).

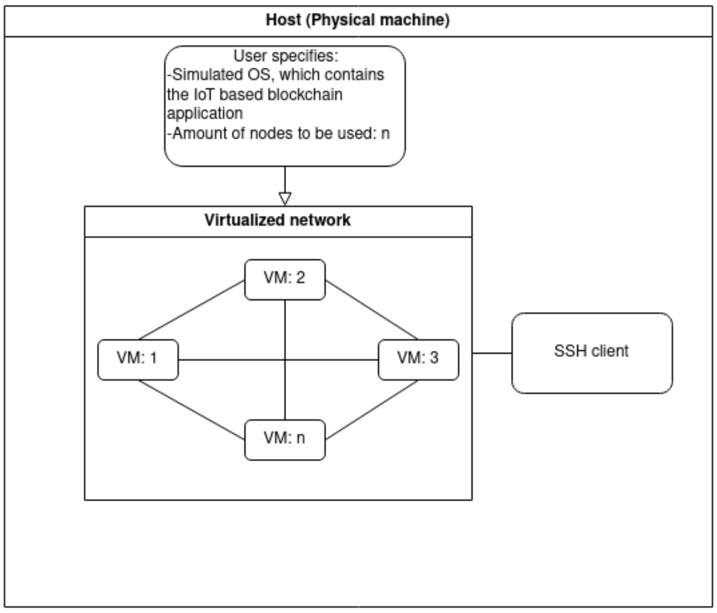
Host (Physical machine)
User specifies:
-Simulated OS, which contains the IoT based blockchain application
-Amount of nodes to be used: n
Virtualized network
VM: 2
VM: 1
VM: 3
VM: n
SSH client

Figure 6. Specification of the testing tool

4.4 Tool Requirements

Blockchain-enabled IoT applications have many different areas which affect their scalability. The first area affecting blockchain-enabled IoT applications' scalability is their network capacity. If a device does not connect to other devices, it cannot function as a part of the chain. Additionally, the amount of devices that are a part of the chain affects the performance of the chain. From a testing environment perspective, this means that at least two aspects of networking should be able to be configured. Firstly, the network connections of devices. This means that in the environment, both connectivity and connection speeds should be able to be configured. Secondly, the number of devices that are connected to the chain. This means that the number of devices that are deployed should be able to be configured when setting up the environment.

Blockchain transactions often require many computationally expensive operations. This can be an issue with IoT devices since they are often restricted by computational power, memory, and networking equipment. Because of this, the testing environment should contain the possibility to simulate hardware used in IoT devices. Additionally, different kinds of hardware should be able to be tested. This could help integrate blockchain-enabled IoT applications since running tests with different hardware configurations could determine a minimum requirement for the application.

Because of the computational limitations of IoT devices, they may require a customized OS to be used. This can lead to security issues if the OS is not configured correctly. Additionally, the IoT devices deployed in a network may not run the same OS. This results in two different requirements for the testbed. Firstly, the testbed must support custom operating systems. This allows the development of the operating system on the IoT devices as a blockchain-enabled IoT application is being developed. Secondly, the testbed must support using various operating systems on IoT devices.

Blockchain-enabled IoT applications can require setup on the IoT devices to run the application. From a testing perspective, the setup of the applications should be automated, allowing the tester to quickly set up a varying amount of connected nodes in the test. Additionally, the software that is on nodes should be able to be configured. This would allow the possibility to test out different applications and their scalability.

Blockchain-enabled IoT applications can utilize many different optimizations. One example of this is to use a server for offloading computations [52, 5]. Another example is offloading off-chain data to a storage server [5, 46, 7]. While testing these applications, there should exist a possibility to configure these types of servers into the network. This would allow the tester to see the impact of adding optimizations to the application. These aspects lead to some requirements for the tool, summarized in Table 5.

Category	**Requirement**	**Reasoning**
Network	Configurable amount of devices	Allows to see the effect of scaling up
	Possibility to add support servers	Allows to test optimisations
	Configurable network connections	Allows simulation of different network configurations
Hardware	IoT hardware simulation	Can help to design and choose the correct hardware for the application
OS	Simulation of custom OSs	Can help to design and choose the correct OS for the application
Application deployment	Automated deployment	Allows scaling up tests

Table 5. Design requirements of the testing tool

4.5 Summary

This Chapter provided an overview of the process of creating the proposed tool. First, the Chapter presented the research method we used in this thesis, which consists of 3 steps: Tool design, Implementation, and Verification and Feasibility. After, we presented the tool design process of this thesis. First, the key stakeholders were identified to be application developers and application testers. The goals of these stakeholders were then identified, and specification for the tool was created from these goals. These specifications resulted in some requirements for the tool. This Chapter focused on addressing $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$ by proposing a way to develop and evaluate a scalability testing tool for blockchain-enabled IoT applications.

5 Implementation of Scalability Assessment Tool

This Chapter describes the implementation process of the scalability assessment tool. Section 5.1 explains why certain virtualization technologies were chosen for the tool. Section 5.2 then describes the implementation by providing the source code and discussing the source code. This chapter is then summarised in Section 5.4.

5.1 Virtualization Software

Blockchain-enabled IoT applications that require testing can be tested in 3 main ways. Firstly, they can be deployed on multiple hardware instances [18, 19, 20]. This is one of the more common solutions to testing these applications. However, this can be expensive to implement since these devices may have a high cost[1] and may require time to configure all of the devices as well as the network of the devices. The second way of deploying applications is to deploy them on virtual machines [65]. This allows testing the application entirely on simulated hardware and software. However, deploying entire VMs is expensive from a computational perspective. The third way of testing blockchain-enabled IoT applications is to use containerization [12]. These containers are isolated environments that share the underlying operating system with each-other [65]. These containers can be run on a shared VM or the physical machine's OS. Application containerization allows more efficient use of resources on the physical computer since each application instance does not have to have its operating system [65]. The difference between deploying applications on VMs and Containerized VMs is presented in figure 7.

[1]A single raspberry Pi's price was close to 45€ `https://www.electrokit.com/produkt/raspberry-pi-3-1gb-model-b/`

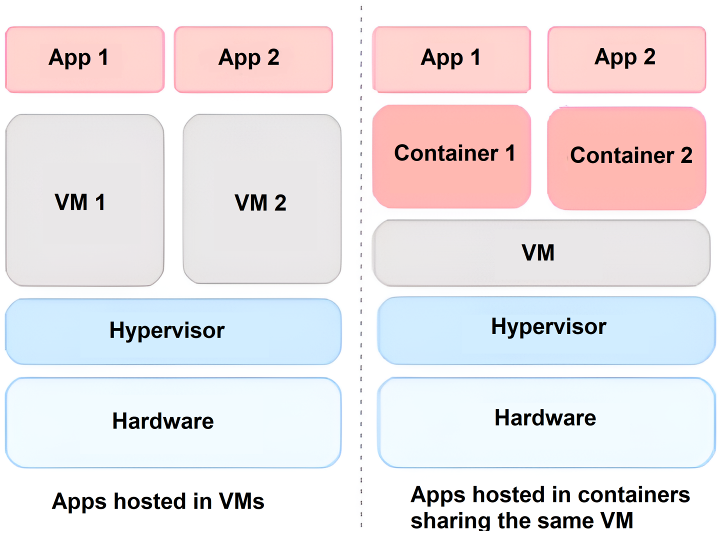

Figure 7. Differences of applications hosted on separate VMs and containerized applications [65].

However, containerization for blockchain-enabled IoT applications is impossible in this testing environment. This is due to the requirements of the testing environment. Since blockchain-enabled IoT applications may face resource constraints, they must be tested with limited hardware. If the applications are containerized, they must share resources on the VM they are running on. This results in the applications potentially having access to more computing resources since the shared VM would need to have more resources. Containerization will work if the main testing purpose is to test the application logic.

Due to these reasons, the virtualization software that is used should allow the creation of virtual machines that use their OS. One prominent tool for this use case is Vagrant [40]. Vagrant is an open-source tool that allows the automated creation of virtual machines. Additionally, it allows the user to choose which VM provider it uses. For example, VirtualBox, Hyper-V, and Docker are officially supported VM providers. Vagrant utilizes many common Command Line Interface (CLI) tools in software development. It can be used to create reproducible virtual testing environments without sharing OS disk images since Vagrant can download the needed OS from the Vagrant cloud, after which modifications can be made locally to that OS. This makes using vagrant ideal, for example, in blockchain-enabled IoT applications' continuous integration pipelines, where environments must be reproducible in cloud environments with repeatable results.

For this environment, the VM provider VirtualBox was chosen. This is due to 2 main reasons. Firstly, VirtualBox is free and open source. This means that if there are issues with the virtualization software, they can be examined through the source code of VirtualBox. The second reason for using VirtualBox is that it allows the user to specify what type of hardware is used. For example, network adapters, CPU cores, memory, and hard disk size are all configurable parameters of VirtualBox VMs. Since IoT devices are limited by hardware, it is necessary to simulate the lack of resources on these devices to examine the application's behavior as resources are constrained.

Simulated Network: Virtualbox allows the user to specify network connections manually. Additionally, it allows the user to create private networks, where NAT is used to connect the private network to the public internet. Vagrant allows the user to specify these network settings when provisioning new VMs. In the proposed tool, a private network model was chosen. In this private network, each node has a direct connection to each other, with the addition of a connection to a gateway that leads to the public internet.

5.2 Tool Content

The development of the tool was started by using the virtual network environment for student projects as a baseline provided by the GitHub page of Professor Tuomas Aura[2].

The virtual tool we implemented uses a combination of Linux shell scripts, python scripts, and vagrant commands. A walkthrough of the code of our tool was presented in a video. The tool has the following file structure:

```
IoTEnvTool/
├── iotBox/
│   ├── buildIotBox.sh
│   ├── setup.sh
│   └── Vagrantfile
├── templates/
│   ├── device.template
│   └── Vagrantfile.template
├── genVagrantFile.py
└── startup.sh
```

The files in the folder `iotBox/` were copied with minimal changes from the folder `base/` in the testbed implemented by Aura [3]. The files inside the folder `iotBox/` are used to build a custom image of an OS that the user specifies. The custom image is then written to a file which can then be utilized to set up other VMs. The shell script `buildIotBox.sh` initiates these actions. The source code of `buildIotBox.sh` was modified from the file

[2]https://github.com/tuomaura/cs-e4300_testbed

[3]https://github.com/tuomaura/cs-e4300_testbed/tree/master/base

base/base.sh[4]. The filename was changed to represent functionality, the names of the outputs were modified, and comments were added to describe what the shell script does. The script does the following:

- Remove previous instances of the packaged OS (if they exist). This is done since this command is only run manually in two situations. The first situation is when the file "iot.box" does not exist in the root folder. The second situation is when the user wants to rebuild the box manually.
- Provision the VM using Vagrant that is specified by the `Vagrantfile` in the folder. This is done to prepare for the next step.
- Package the OS into a file and add it to a variable in Vagrant. This is done to speed up the creation of the environment.
- Destroy the running VM and clear the local cache of Vagrant in the current folder. This is done since these are not needed anymore.

This allows the user to specify hardware aspects of a common device and install common software to the device, which will then be copied into multiple instances in the final environment. Additionally, it can be used as a debugging environment for a singular device. The modified source code of `buildIotBox.sh` is included in Listing 1.

```
# Remove previous build
vagrant destroy -f iotBox
vagrant box remove iotBox
# Provision iotBox
vagrant up iotBox --provision
# Package the provisioned VM
vagrant package --output ../iot.box
# Add box to vagrant
vagrant box add iotBox ../iot.box -f
# Destroy running VM
vagrant destroy -f iotBox
# Remove vagrant cache
rm -rf .vagrant/
```

Listing 1. `buildIotBox.sh` source code

The shell script `startup.sh` handles the user input and starts the tool. The processing logic of the script has been modeled as a flowchart in Figure 8.

[4]https://github.com/tuomaura/cs-e4300_testbed/blob/master/base/base.sh

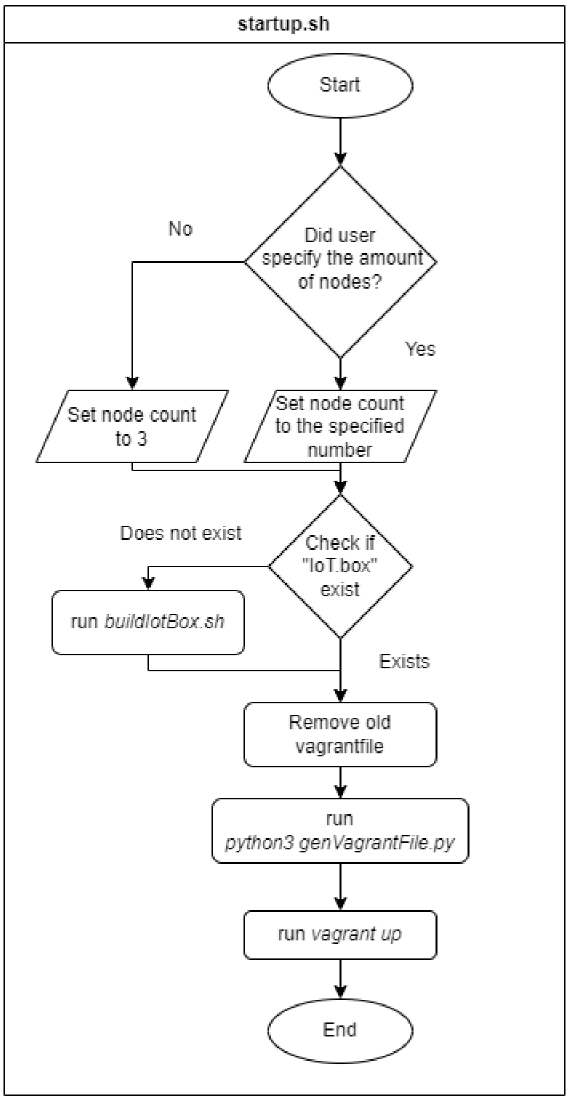

Figure 8. Processing logic of startup.sh

The script starts by analyzing the input parameter n. This parameter represents the number of IoT devices the user wants to create. The script will default to 3 nodes if this parameter is not provided. After this, the script checks if "iot.box" file exists. If not, it will run the previously mentioned `buildIotBox.sh` shell script and build the base box. After the box has been built, the script checks if a previous Vagrantfile exists and will remove it. This makes it possible to quickly run different environments with different amounts of IoT devices, to test the impact of additional nodes. After removing the old Vagrantfile, the script runs the Python script `genVagrantFile.py` with the number of devices specified by the input parameter n. This Python file generates the resulting Vagrantfile for the environment. After the file has been generated, the script runs the command Vagrant up, which builds the testing environment. The source code of startup.sh is presented in Listing 2.

```
# Get arg 'n', set default to 3 if not provided
num=3
while getopts n: flag
do
    case "${flag}" in
        n) num=${OPTARG};;
    esac
done
# Check if prebuilt box exists
BOX=iot.box
if test -f "$BOX"; then
    echo "$BOX exists."
else
    # Create prebuilt box
    cd iotBox
    ./buildIotBox.sh
    cd ..
fi
# Check if previous environment exists and destroy if it exists
FILE=Vagrantfile
if test -f "$FILE"; then
    echo "$FILE exists, removing and destroying previous env"
    vagrant destroy -f
    rm Vagrantfile
fi
# Generate new vagrantfile
python3 genVagrantFile.py $num
# Run vagrant
vagrant up
```

Listing 2. startup.sh source code

Creating the Vagrantfile with the genvagrantfile.py-script is generated by combining two templates. The first template is the Vagrantfile template. This template contains the necessary configurations for Vagrant and the string "#Start", which indicates the starting position for the Python script. Within the Vagrantfile template, the user can also specify other custom virtual machines that can be used in the environment. This allows users to test optimizations, such as a computation offloading server. An example of this template is presented in Listing 3.

```
Vagrant.configure("2") do |config|
config.vm.define "bootNode" do |bootNode|
  bootNode.vm.box = "iotBox"
  bootNode.vm.network "private_network",
    ip: "10.1.0.2",
    netmask: "255.255.255.0",
    virtualbox__intnet: "private_net"
  bootNode.vm.hostname = "bootNode"
  bootNode.vm.provision :shell, path: "scripts/setupBootNode.sh",
   env: {"IPADDRESS" => "10.1.0.2"}
  bootNode.vm.provider "virtualbox" do |vb|
    vb.name = "bootNode"
    vb.cpus = 1
    vb.memory = 1024
  end
end
  #Start

end
```

Listing 3. Vagrantfile.template example

The second template file is the device template file. This template allows the user to specify what OS the IoT devices will use (by default, the "base.box" will be used) and to specify commands to be run by the devices. An example of this device template is presented in Listing 4.

```
config.vm.define "deviceX" do |deviceX|
  deviceX.vm.box = "iotBox"
  deviceX.vm.network "private_network",
    ip: "10.1.0.1SUBADDR",
    netmask: "255.255.255.0",
    virtualbox__intnet: "private_net"
  deviceX.vm.hostname = "deviceX"
  deviceX.vm.provision :shell, path: "scripts/startup.sh" , env: {"
   IPADDRESS" => "10.1.0.1SUBADDR", "NAME" => "deviceX"}
  deviceX.vm.provider "virtualbox" do |vb|
    vb.name = "deviceX"
    vb.cpus = 1
    vb.memory = 1024
    vb.customize ["modifyvm", :id, "--cpuexecutioncap", "50"]
  end
end
```

Listing 4. device.template example

These templates are used by the `genVagrantFile.py` script. The `genVagrantFile.py` Python script inserts the string contained in the device template n amount of times into the Vagrantfile template. While doing it, the script makes two modifications to the template. The first modification that is done is replacing the device names. The template has the device name `"deviceX"`, which is replaced in the Vagrantfile with "device" + the index of the device. For example, the first device will be named *"device1"*, and the second device will then be named *"device2"*. This allows the creation of multiple similar devices with different names in Vagrant. The second modification made to the device template is the device's IP address. The script searches for the string *"SUBADDR"* and replaces it with the index. This results in a unique private IP address for each device. For example, if the IP address in the device template is defined as *"192.168.58.SUBADDR"* the result for the first device would be *"192.168.58.1"*. This is because it allows the user to avoid manual network configuration and can instead automatically set the IP addresses of the devices. The processing logic of the script has been modeled as a flowchart in Figure 9, and the file's source code is presented in listing 5.

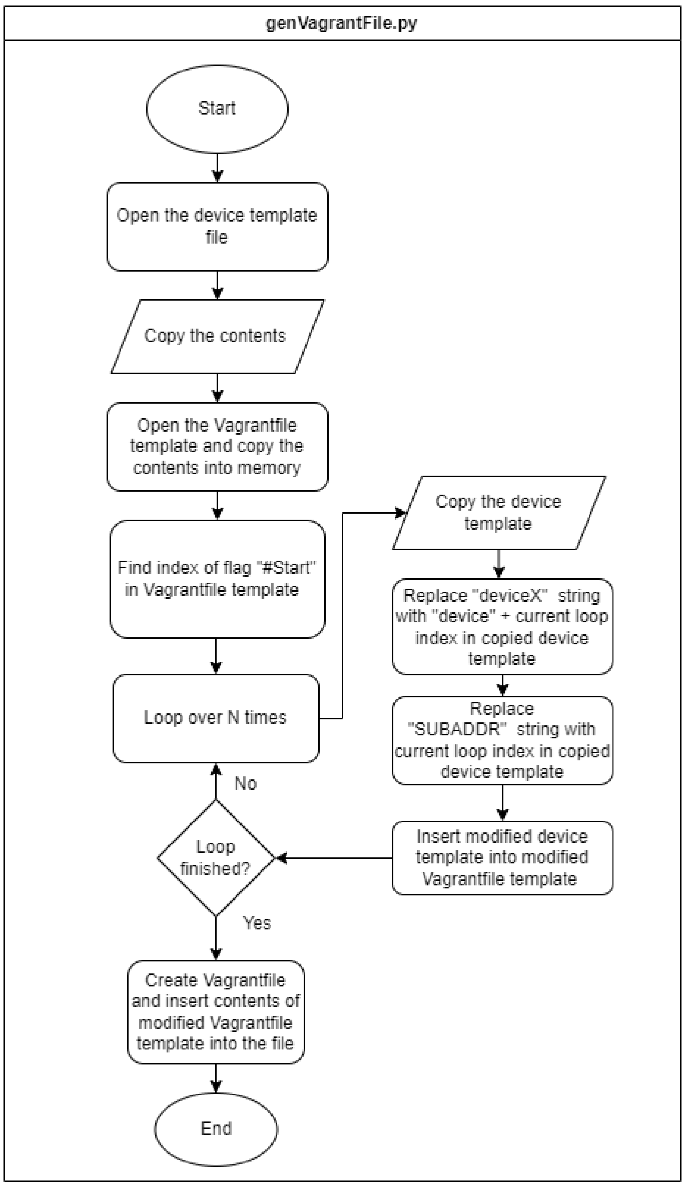
genVagrantFile.py
Start
Open the device template file
Copy the contents
Open the Vagrantfile template and copy the contents into memory
Find index of flag "#Start" in Vagrantfile template
Loop over N times
Copy the device template
Replace "deviceX" string with "device" + current loop index in copied device template
Replace "SUBADDR" string with current loop index in copied device template
Insert modified device template into modified Vagrantfile template
Loop finished?
No
Yes
Create Vagrantfile and insert contents of modified Vagrantfile template into the file
End

Figure 9. Processing logic of genVagrantFile.py

```
#!/usr/bin/env python3
import sys
# Number of iot devices
deviceAmount = int(sys.argv[1])
# Get device template string
deviceTemplateFile = open('templates/device.template', 'r')
lines = deviceTemplateFile.readlines()
deviceTemplateFile.close()
```

```python
deviceTemplate = ''.join(lines)
# Get vagrantfile template
vagrantfileTemplateFile = open('templates/Vagrantfile.template', 'r')
lines = vagrantfileTemplateFile.readlines()
vagrantfileTemplateFile.close()
vagrantfileTemplate = ''.join(lines)
# Generate vagrantfile
vagrantfile = vagrantfileTemplate
i = vagrantfile.find("#Start") + 7
for deviceNum in range(1, deviceAmount + 1):
    addedDevice = deviceTemplate.replace("deviceX", "device" + str(
    deviceNum)).replace("SUBADDR", str(deviceNum))
    vagrantfile =  vagrantfile[:i] + addedDevice+ vagrantfile[i:]
    i+= len(addedDevice)
# Write to file
vagrantfileOut = open('Vagrantfile', 'w')
vagrantfileOut.write(vagrantfile)
vagrantfileOut.close()
```

Listing 5. `genvagrantfile.py` source code

5.3 Limitations of the Tool

Due to the chosen underlying frameworks, the tool has some limitations. The first limitation is the number of devices that can be created with the tool. The amount of memory that the host of the testing environment requires must be greater than the amount of memory assigned to all the virtual machines. This creates a hard limit for the number of devices that can be deployed. Another possible limiting factor is the disk space of the host. If the VMs are deployed in a way that allocates disk space directly from the hard drive (and not using dynamic storage allocation), it can limit the number of devices that can be deployed.

Another limiting factor is the number of CPU cores on the host. VirtualBox assigns physical CPU cores to each VM. This limits the number of VMs that can be running at one time. This limits the number of simulated devices to the number of cores the host has if each of the VMs is running on a single core. VirtualBox allows setting an execution cap to assigned cores. This execution cap limits the time the physical CPU processes the virtual CPU. This can be utilized to limit the processing power that the VMs have. Thus, setting an execution cap results in the VM having a lower clock speed.

5.4 Summary

This Chapter presented the implementation of the proposed tool. First, we explained the choice of underlying technologies used in the tool. These included using Vagrant, VirtualBox, and Python. Then we presented how the tool was implemented. The tool combined these previous technologies and shell scripting to implement the proposed tool. Then we discussed the potential limitations of using the tool based on the limitations of the underlying technology. This Chapter continued addressing **RQ**$_2$ by providing an implementation for the provided tool.

6 Validation and Results

This Chapter describes the validation process we used to estimate the feasibility of our tool. These tests aim to validate the use of the tool in assessing the scalability of blockchain-enabled IoT applications. Our validation method is the following. First, we establish common variables for testing. This means that the non-scaling factors and nuisance variables are set to be static. Discussion about these variables is presented in Section 6.1. Then, after establishing common variables for the tests, the basic functionality is validated by running a blockchain network on simulated IoT devices using the tool. This is done to verify that the tool can be used for running custom blockchain networks and to validate that the tool's basic functionality works as expected. This test is described in Section 6.2. The second validation test is then conducted to validate the use of the tool in assessing the scalability of blockchain-enabled IoT applications. This test is conducted to estimate the feasibility of using our tool in assessing the scalability of blockchain-enabled IoT applications. Section 6.3 discusses this test and its results.

6.1 Common Variables for the Tests

As mentioned in the background, the scalability of blockchain-enabled IoT applications consists of 3 main factors: scaling factors, non-scaling factors, and nuisance variables. In this Section, we establish common variables for the following non-scaling factors: the consensus mechanism of the underlying blockchain, the type of (simulated) IoT device, and the OS used on nodes. Processor speed as a nuisance variable is also discussed.

The `device.template` file contains the hardware properties set by VirtualBox for each device with the proposed tool. Previous works have utilized different versions of the Raspberry Pi [18, 19, 20]. For example, Wang et al. [19] used a Raspberry Pi 3 Model B to conduct their experiment. The Raspberry Pi 3 Model B has a Quad-core CPU with a clock speed of 1.2GHz and 1 GB of ram [5]. For these tests, the IoT devices were chosen to have a single CPU core, which was limited to running only 10% of the time of the host machines clock speed. The reason for choosing only 1 CPU core is that it allows the creation of more IoT devices with a small number of physical cores. For example, the AMD Ryzen 7 7300X only contains eight cores, thus limiting 2 IoT devices if the core count for each node is 4. The simulated CPU speeds were set to 10 % to amplify

[5]`https://www.raspberrypi.com/products/raspberry-pi-3-model-b/`

the impact of requests since less processing power should lead to increased impacts in request times.

The IoT devices need an OS to set up and run the blockchain-enabled IoT applications. Previous works have used both Raspbian OS and Ubuntu on IoT devices as the operating system [18, 19, 20]. For this testing environment, the Ubuntu Core operating system was first chosen to be tested. This is because the OS is tailored to run on IoT devices with limited hardware, such as the Raspberry Pi. However, importing the operating system for this testing type was unsuitable for two reasons. Firstly, the OS does not support inserting private SSH keys into the system. The OS uses an email address to fetch a public SSH key, which is then used for SSH authentication. This makes vagrant configuration complex. The second issue with the OS running on VirtualBox was that it defined the disk space as read-only. This means that no additional software could be installed on the device itself. After Ubuntu core was deemed unsuitable for the test, Ubuntu 18.04.6 server was chosen as the OS. There were four main reasons for choosing Ubuntu 18.04.6 as the OS for the nodes. Firstly, the OS was used by previous works [18, 19, 20]. This does not guarantee that the OS is the most suitable for the IoT devices, but it allows to compare differences between the experiments. The second reason for choosing the OS is that it is easy to deploy. Vagrant can download OS disk images directly from the cloud without the user configuring anything manually. This allows setting up the OS without distributing the modified disk image. The third reason for choosing the OS is that Ubuntu 18.04.6 is well-documented, and configuration errors are well-documented with virtual machines running the OS. This speeds up the configuration time and debugging of the environment. In addition, the OS includes a package manager and many basic tools used in software development, such as OpenSSL and Python. Finally, the OS meets the recommended requirements of the simulated device hardware. The recommended hardware for an Ubuntu server is a CPU with a clock speed of 1 GHz or better and 1 GB of RAM [6], which these simulated devices match. The OS is downloaded, modified, and packaged before setting up the environment. This is done to speed up the creation of the environment. If the OS were downloaded and modified for each device separately, it would increase the creation time, especially in cases where many devices are created. The modifications to the OS and the packaging process can be seen in both tests in the `iotBox` folder.

6.2 Validation Test 1: Basic Functionality

This section presents the validation test conducted to validate the basic functionality of the implemented tool. During this test, some additional tools were developed to assist with configuring the tested custom Ethereum network. These additional tools are in a separate branch of the tool's repository. The source code that was used in the tests with

[6] `https://ubuntu.com/server/docs/installation`

the tools can be found on the validationTest1 branch of the GitHub repository

6.2.1 Testing scenario

In this testing scenario, the tool was tested to validate the tool's basic functionality by using the tool to deploy and run Geth clients on simulated IoT devices. Previous works in testing blockchain-enabled IoT applications have utilized the Geth-client [19, 20, 12]. Of these, only PlaTIBART [12] utilized virtualization for testing. The Geth client was chosen as the tested application for three main reasons. Firstly, it has been used widely in previous works. The second reason for choosing the Geth client is that it is a versatile application. It allows the user to interact with a private Ethereum chain in multiple ways through the host's CLI. The third reason for testing the Geth client is that it is a widely documented and deployed application[7]. This can help to speed up the configuration of the client since resources are widely available.

The clients generated in this test use a private Ethereum chain that uses the Ethash consensus method. This consensus method is a PoW consensus method, which is not optimal for IoT devices. Ethereum uses a PoS consensus method since the blockchain was split into two chains. However, using the PoS consensus method requires additional software for the nodes. The current Ethereum PoS consensus requires a consensus and an execution client, such as the Geth client. These consensus clients require much memory and processing power to function properly. For example, the recommended hardware requirement for the Lighthouse consensus client is a quad-core CPU with 32GB of RAM. In addition, previous works in testing blockchain-enabled IoT applications have used the pre-merge Ethereum chain with the Ethash consensus method. This allows a comparison of these previous works to this application.

6.2.2 Environment Setup

The implemented tool does not provide configuration for device wallet addresses, private keys, or the genesis block of the custom Ethereum chain. Thus helper tools were developed to help configure these parameters for the validation test.

To deploy the Geth client, the VMs need to install the client, unlock their wallet and run the client. To do this, the commands needed to install the Geth client were placed in the previously mentioned `ioTBox/setup.sh`-file as follows:

```
sudo apt-get update -y
sudo apt upgrade -y
wget https://gethstore.blob.core.windows.net/builds/geth-linux-amd64
    -1.11.6-ea9e62ca.tar.gz
tar xvzf geth-linux-amd64-1.11.6-ea9e62ca.tar.gz
```

[7]Geth is the most deployed execution client on the Ethereum main net as of May 8th, 2023 https://clientdiversity.org/

```
cd geth-linux-amd64-1.11.6-ea9e62ca/
sudo chmod +x geth
sudo cp geth /usr/local/bin/
```

Listing 6. The modified `ioTBox/setup.sh`

Installing the client in the packaging phase will speed up the environment creation since all the devices participating in the test will utilize the client. However, from this file, it is impossible to launch the application since that requires individual parameters related to the VMs, such as the wallet address of the VM.

To help set up and run the application, the `scripts/setupDevice.sh` was created. The contents of the script are presented below:

```
geth init --datadir node /vagrant/secrets/genesis.json
geth account import --datadir node/ --password /vagrant/secrets/pw.
    txt /vagrant/secrets/keys/$HOSTNAME

tmux new-session -d -s my_session 'ENODE=`cat /vagrant/secrets/
    bootnodeconf/enodeaddr` && WALLET=`printf "0x%s" "$(cat /vagrant/
    secrets/addresses/$HOSTNAME)"`&& sudo geth --datadir node --port
    30306 --bootnodes $ENODE --networkid 12345 --authrpc.port 8551 --
    unlock $WALLET --password /vagrant/secrets/pw.txt'
```

Listing 7. The `scripts/setupDevice.sh` script

The script does the following:

- Initialize a private Ethereum chain into the folder node, from a genesis configuration file `/vagrant/secrets/genesis.json`
- Import a private key into the client from the text file `/vagrant/secrets/keys/$NAME` and use the password provided in the textfile `/vagrant/secrets/pw.txt` to protect it. In this case, the reason for using a password is that the Geth client does not allow importing accounts without authentication.
- Start a detached shell session, where the Geth client connects to the Ethereum network with the id:12345 and uses the data directory that was previously initialized for the chain. Additionally, the command adds a boot node address to connect the peers. Additionally, the command unlocks the wallet of that specific device, which allows it to send transactions from that wallet.

The `scripts/setupDevice.sh` script requires some configuration files to be created before it can be used with the VMs. These files can be manually configured, but a helper script was created `genSecrets.sh`, which automatically generates these files. This script generates three different configuration files. The script is listed below:

```
num=3
# Get the input variable n, if it exists
while getopts n: flag
do
    case "${flag}" in
        n) num=${OPTARG};;
    esac
done
# Copy the template for the genesis template
cp ../templates/genesis.json.template genesis.json

for i in $( seq 1 $num )
do
    # Generate a private key
    openssl ecparam -name secp256k1 -genkey -noout | openssl ec -text -noout > key
    # Extract the private key, remove the leading zero byte and pipe it into a file
    cat key | grep priv -A 3 | tail -n +2 | tr -d '\n[:space:]:' | sed 's/^00//' > priv
    # Generate a wallet address based on the private key, and write it into a file
    python3 genPubAddr.py
    # Move the generated private key to correct location
    mv priv keys/device$i
    # Add balance to the generated address
    printf "        \"%s\" : { \"balance\": \"5000000000000000\"},\n" "$(cat addr)" >> genesis.json
    # Move wallet address to correct location
    mv addr addresses/device$i
done
# remove the temporary key
rm key
```

Listing 8. The `secrets/genSecrets.sh` script

The script does the following actions

- Firstly, it generates n amount of Ethereum private keys using OpenSSL, where n is an input parameter with a default value of 3. These keys are then piped to files which are named after the devices that are generated.
- After, the script generates wallet addresses for each device's private keys. This is done using the python3 script `secrets/genPubAddr.py`.
- Finally, the script generates the `genesis.json` file, which is used to initialize the private Ethereum chain, mentioned previously. The script assigns 5000000000000000 Wei by default to each of the generated accounts. The `genesis.json` is generated

by appending account balances to the `templates/genesis.json.template` file. This means that the genesis block of the chain can be re-configured by either modifying the genesis.json file or by modifying the template file and then running the `genSecrets.sh` script.

The python script `secrets/genPubAddr.py` mentioned takes an Ethereum private key file and uses the eth-keys python module to generate a wallet address for that key. This address is then piped into a file, which holds the address for the corresponding private key. The source code of the script is listed below:

```python
    from eth_keys import keys
    from eth_utils import decode_hex
    # Open the private key file
    pkFile = open('priv', 'r')
    # Read contents, strip line endings and close the file
    pkString = pkFile.readlines()[0].strip()
    pkFile.close()
    # Create byte array from the string
    pkBytes = bytearray.fromhex(pkString)
    # Use eth_keys to generate the public key
    pk = keys.PrivateKey(pkBytes)
    # Write public key to file
    addrFile = open('addr', 'w')
    addrFile.write(pk.public_key.to_checksum_address()[2:])
    addrFile.close()

```

Listing 9. The `secrets/genPubAddr.py` script

After these configuration files were complete, the templates provided by the tool needed to be configured. For this test, the `templates/Vagrantfile.template` was modified to contain a single static node. This node is the boot node for the private Ethereum peers. This means the node does not interact with the chain but instead connects peers. The boot node is required since the deployed blockchain is private. The boot node is assigned a private IP address 10.1.0.2, from which it hosts the bootstrapping server. The node is running the same simulated OS and hardware as the other devices in the network. `Vagrantfile.template` used for this test is provided below:

```ruby
Vagrant.configure("2") do |config|
config.vm.define "bootNode" do |bootNode|
  bootNode.vm.box = "iotBox"
  bootNode.vm.network "private_network",
    ip: "10.1.0.2",
    netmask: "255.255.255.0",
    virtualbox__intnet: "private_net"
  bootNode.vm.hostname = "bootNode"
  bootNode.vm.provision :shell, path: "scripts/setupBootNode.sh",
   env: {"IPADDRESS" => "10.1.0.2"}
  bootNode.vm.provider "virtualbox" do |vb|
    vb.name = "bootNode"
    vb.cpus = 1
    vb.memory = 1024
  end
end
  #Start
end

```

Listing 10. The used `Vagrantfile.template` template

The boot node is configured to run the `scripts/setupBootNode.sh` script once Vagrant starts the device. This script handles the application setup for the bootstrapping. The script is listed below:

```
# Download and install the bootstrapping application
wget https://gethstore.blob.core.windows.net/builds/geth-alltools-
    linux-amd64-1.11.6-ea9e62ca.tar.gz
tar xvzf geth-alltools-linux-amd64-1.11.6-ea9e62ca.tar.gz
cd geth-alltools-linux-amd64-1.11.6-ea9e62ca/
sudo chmod +x bootnode
sudo cp bootnode /usr/local/bin/
# Start bootstrapping application
tmux new-session -d -s my_session 'bootnode -nodekey /vagrant/secrets
    /bootnodeconf/boot.key -addr "10.1.0.2:30305"'

```

Listing 11. The `scripts/setupBootNode.sh`-script

The configuration file `/vagrant/secrets/bootnodeconf/boot.key` was manually generated and included in the repository. This parameter was manually generated since the node is static in all the tests.

The final modifications used in these tests were made to the `templates/device.template`, which contains the Vagrant configuration for each device. The previous section 6.1 explains both the simulated hardware and the OS used in the devices. The contents of this template are listed below:

```
config.vm.define "deviceX" do |deviceX|
  deviceX.vm.box = "iotBox"
  deviceX.vm.network "private_network",
    ip: "10.1.0.1SUBADDR",
    netmask: "255.255.255.0",
    virtualbox__intnet: "private_net"
  deviceX.vm.hostname = "deviceX"
  deviceX.vm.provision :shell, path: "scripts/startup.sh" , env: {"
    IPADDRESS" => "10.1.0.1SUBADDR", "NAME" => "deviceX"}
  deviceX.vm.provider "virtualbox" do |vb|
    vb.name = "deviceX"
    vb.cpus = 1
    vb.memory = 1024
    vb.customize ["modifyvm", :id, "--cpuexecutioncap", "10"]
  end
end

```

Listing 12. The `templates/device.template` template

Each device is added to a private network. The IP address of each device in the private network is 10.1.0.1SUBADDR, where the device number replaces the string SUBADDR. For example, device one would have the address 10.1.0.11 in the final Vagrantfile. This private network is used to run the custom Ethereum chain. The template also defines a startup script for the devices run after Vagrant has started each device. This script handles the application's setup on the devices and starts the application. The script specifies each node's IP address and device name as an environment variable.

6.2.3 Blockchain Setup

The Geth client requires a bootstrapping node to function. This is because Ethereum peers often are behind a NAT network and a firewall, which does not allow exposing ports to the public internet. The bootstrapping nodes are pre-defined in the Geth client for the main and common testing networks. However, these do not work with custom testing networks, such as the one used in this environment. The boot node in the network is set up by the command `bootnode -nodekey /vagrant/secrets/bootnodeconf/boot.key -addr "10.1.0.2:30305"` . This starts the boot node with the specified key on port 30305. However, since this command starts a server, the process does not exit, which hangs the setup in Vagrant. For this reason, the command line program tmux is used to create the process and leave it running, while Vagrant sets up the rest of the environment. The full command that is ran on startup is `tmux new-session -d -s nodeSession 'bootnode -nodekey /vagrant/secrets/bootnodeconf/boot.key -addr "10.1.0.2:30305"'`

The devices that run the Geth client require some more setup. Firstly, the Geth client needs to initialize the custom Ethereum chain. This is done by the command `geth init -datadir node /vagrant/secrets/genesis.json`. After this, the IoT device can import an Ethereum account into the Geth client. The command used for this is `geth account import -datadir node/ -password /vagrant/secrets/pw.txt /vagrant/secrets/keys/$NAME`. The script imports an account from the key file `/vagrant/secrets/keys/$NAME` (these keys were generated with the `genSecrets.sh` script) into the Geth client and sets a password to the account, which is equal to the contents in `/vagrant/secrets/pw.txt`. The variable $NAME is the environment variable passed by Vagrant to the script. For example, for device 1, the environment variable $NAME would be `device1`. After importing the account into the client, the startup script sets two environment variables: ENODE and WALLET. The

ENODE variable is the file's contents in /vagrant/secrets/bootnodeconf/enodeaddr. This contains the full address of the boot node, which is hard-coded into the configuration. The WALLET variable is set by concatenating the string "0x" with the contents in the file /vagrant/secrets/addresses/$NAME, which contains the wallet address for each device (these addresses were generated with the `genSecrets.sh` script). Once these variables are set, the command `tmux new-session -d -s nodeSession ' sudo geth -datadir node -port 30306 -bootnodes $ENODE -networkid 12345 -authrpc.port 8551 -unlock $WALLET -password /vagrant/secrets/pw.txt'` can be run. This command utilizes the tmux command line program to create a detached session for the Geth client. The Geth client is started with the custom chain initialized in the node directory. The command sets the discovery port to 30306 and manually adds the boot node's address by utilizing the environment variable $ENODE. The command defines the network id as 12345 and enables the authrpc server to be run on the localhost with port 8551. Finally, the command unlocks the wallet belonging to the device, allowing transactions to be sent from that wallet.

6.2.4 Testing Method

To validate the basic functionality of the tool, the tool should be able to set up environments where a blockchain network functions without errors. Additionally, using the tool should be stable and repeatable.

To test these properties, we used the previously mentioned configuration to set up an environment with 2,4,6,8, and 10 nodes. From these setup runs, the following properties were measured:

- The time to create the environment
- The number of peers connected to the private chain
- Amount of times that an example transaction was successfully sent from the Geth-client

The tests were run on Ubuntu 20.04.6 with 32GB of RAM and the AMD Ryzen 3700X CPU running at the base clock of 3.2Ghz and boost clocks enabled. These measurements were repeated five times.

The tests were conducted using another shell script named `testTool.sh`. The shell script is listed below:

```
# Copy tool contents to a temporary testing folder
cp -R IoTSimulation/ tempFolder/
cd tempFolder/secrets/
# Generate configuration files
time ./genSecrets.sh -n 2 > ../../genTime
cd ..
```

```
# Run the tool, measure the time to complete and save the time in a
    file
time ./startup.sh -n 2 > ../setupTime
# Query the peers connected to the first device
vagrant ssh device1 -c "sudo geth attach  --exec admin.peers node/
    geth.ipc"
# Send a transaction from the first device
vagrant ssh device1 -c "sudo geth attach --exec \"eth.sendTransaction
    ({to: '0x6c4fe784Cb79502C784521020CD283Cc86E43D42', from: eth.
    accounts[0], value: 25000})\" node/geth.ipc"
# Cleanup
vagrant destroy -f
cd ..
rm -rf tempFolder/
```

Listing 13. The `secrets/genPubAddr.py` script

The script does the following actions:

- Copy the directory containing the tool where the `iot.box` file has been built. The reason for excluding the time to build the OS is that the OS can be copied and remain unchanged while developing an application.
- Generate the required configuration files using the `secrets/genSecrets.sh` script.
- Use the time command to capture the execution time of `startup.sh`, which brings up the environment and starts the Geth client on the devices. Once the `startup.sh`
- Query the first device to see what clients are connected to the device. This command queries the Geth client for connected peers. This command will return a JSON object and print it to the current shell. The output of this command was used to confirm that all created devices were successfully connected to the p2p network.
- Send an example transaction to a static address from the first device. This command uses the Geth client to create a transaction, which transfers funds from device 1 to a hard-coded address. The output will be printed to the shell, where the transaction creation can be verified.

6.2.5 Results

The tests were conducted to verify that the tool's basic functionality works as expected, such as generating devices correctly and automatically starting these devices to attach to the blockchain network. We verified three different properties. The first property is to verify that Vagrant can generate all nodes successfully. This was verified by checking if the script produced any errors. The second property is to check if all peers have connected to the network. This can be verified via the returned JSON string in the shell. The final property that is checked is that transactions can be sent. The transaction can be verified by examining the returned string from the transaction command.

Device count	**Amount of times environment was created succesfully**	**Amount of times all peers were connected**	**Amount of times example transaction was sent successfully**	**Mean time for environment startup (s)**
2	5/5	5/5	5/5	161.0
4	5/5	5/5	5/5	272.2
6	5/5	5/5	5/5	369.0
8	5/5	5/5	5/5	510.9
10	5/5	5/5	5/5	629.6

Table 6. Testing results of the environment on a Ryzen 7 3700X CPU with 32GB of RAM

The test results of the environment are presented in Table 6. All of the tests were completed successfully in this scenario. However, while developing the environment, sometimes the script `genSecrets.sh` would fail. Its failure was because sometimes the `OpenSSL` command used to generate private keys would sometimes generate a private key, where the leading bytes are set to 0. This would lead to the hex value being shorter than 32 characters, which would cause the address generation to fail. This bug in the helpers could be fixed to make the environment setup more stable in these tests.

These tests showcased that the basic functionality of setting up a private blockchain with a variable number of simulated IoT devices is possible with the provided tool. Since these tests were successful, we decided to move on to the second validation test, where the tool is used for scalability testing.

6.3 Validation Test 2: Using the Tool to Assess Scalability

This section presents the validation test conducted to evaluate whether the provided tool can be used to assess the scalability of blockchain-enabled IoT applications. The source code that was used in the tests can be found on the validationTest2 branch of the GitHub repository (`https://github.com/SanttuSi/IoTSimulation/tree/validationTest2`).

6.3.1 Tested Application

One recent use case for blockchain is to use it in the IoV applications [3]. IoV applications must communicate with the infrastructure and other vehicles surrounding them to allow functional operation. However, this can be challenging since vehicle manufacturers create different vehicle types, resulting in the need for a communication standard. In this case, blockchain could be utilized to verify interactions in the physical world between the IoV and verify communication between the vehicles. IoV may contain computers that have enough computing power to complete blockchain operations. However, these IoV networks may contain some limited IoT devices. For example, a traffic light may contain a simple IoT device that communicates with the vehicles near it. This can create a situation where the scale of these applications needs to be assessed.

A proof-of-concept (POC) application is developed, which is then used to assess its scalability and used to evaluate the feasibility of the provided tool. The goal of the POC application is to store vehicle location data on the Ethereum chain. Other vehicles could then use this location data for other applications, such as autonomous vehicles, parking detection, and road toll detection. The application consists of 3 main components: The first component is a vehicle client. The vehicle client does not mine the chain but instead sends transactions to be processed to the chain based on its location. Additionally, the vehicle can confirm the positions of other vehicles it detects. The vehicle client requires specific properties to function with the application:

- A general-purpose computer that can run the application.
- Internet connectivity, e.g., an LTE connection, is used to communicate with the Ethereum chain.
- WiFi connectivity is used to communicate with other vehicles/sensors.

The vehicle client application uses WiFi to find sensors around it. Once the client finds a sensor, it sends the Ethereum address of itself to the peer, to which the peer responds with the address of itself. Then, the client creates a transaction to the Ethereum chain, where the transaction's extra info contains the vehicle's position. The actions done by the vehicle client are modeled in Figure 9. 10.

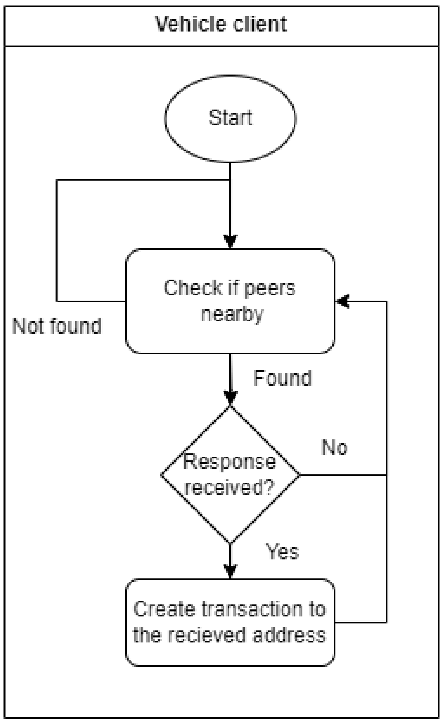

Figure 10. Vehicle client processing logic

The second component is a client for IoT devices. This can be, for example, a traffic light or a sensor on a traffic sign. These devices are the most limited in computing power. The IoT device client requires specific properties to function with the application:

- A general-purpose computer that can run the application.
- Internet connectivity that is used to communicate with the Ethereum chain.
- WiFi connectivity that is used to communicate with vehicles.
- Geopositional location of the client.

The IoT device client hosts a server on the WiFi network that it is connected to. Once it receives an application request via WiFi, it reads the message and creates a transaction to the ETH address contained in the message. It then responds to the request with its own ETH address. The processing logic of the IoT-device client is presented in figure 11.

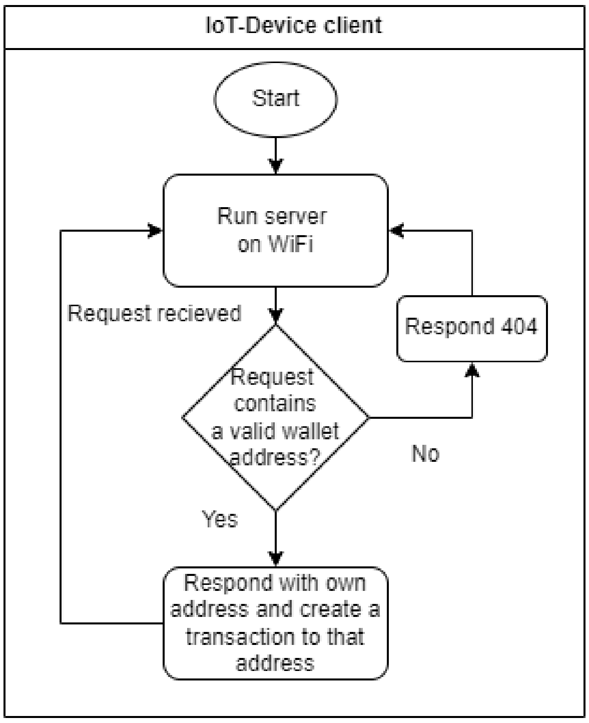
IoT-Device client
Start
Run server on WiFi
Request recieved
Request contains a valid wallet address?
Respond 404
No
Yes
Respond with own address and create a transaction to that address

Figure 11. Sensor client processing logic

The third component of the application is a miner. These miners are static servers that mine the chain to confirm blockchain transactions. The entities that mine the chain are motivated by the rewards that mining produces and do not require any additional software but only require software used to mine the Ethereum chain.

The application is presented as a BPMN in figure 12. In this figure, the vehicle is the activator of the process. It detects a sensor via WiFi and then sends the wallet address to the sensor via WiFi. The sensor then checks if the address is valid and responds to the vehicle with its wallet address. Then, both The sensor and the vehicle create a transaction where they transfer funds to each other. The miners in the network then validate these transactions.

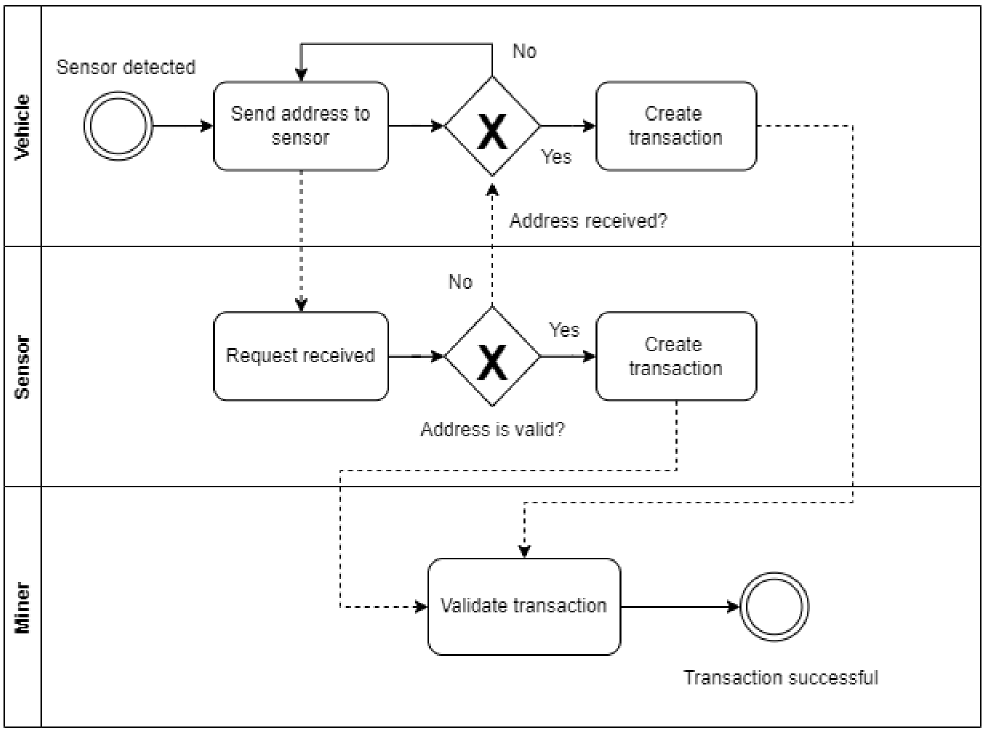

Figure 12. IoV POC application BPMN

Since the application is a POC application, all components are not tested. Instead, only the scalability of the sensor application is tested. The source code of the application is presented below.

```
from http.server import BaseHTTPRequestHandler, HTTPServer
import time
import json
import os
hostName = "0.0.0.0"
serverPort = 8080
class IoTClient(BaseHTTPRequestHandler):
    def do_POST(self):
        try:
            content_length = int(self.headers['Content-Length'])
            body = self.rfile.read(content_length)
            json_object = json.loads(body)
            vehicle_address = json_object["address"]
            command = "sudo geth attach --exec \"eth.sendTransaction
    ({to: \'" + vehicle_address +"\', from: eth.accounts[0], value:
    2500})\" /home/vagrant/node/geth.ipc"
            os.system(command)
            self.send_response(200)
            self.end_headers()
            self.wfile.write(bytes("0
x634473DCE0d0e6e6EDD1c32e34f51f45fe4D2c42", 'utf-8'))
        except:
            self.send_response(404)
            self.end_headers()
webServer = HTTPServer((hostName, serverPort), IoTClient)
print("Server started http://%s:%s" % (hostName, serverPort))
try:
    webServer.serve_forever()
except KeyboardInterrupt:
    pass
webServer.server_close()
print("Server stopped.")
```

Listing 14. The sensor client

6.3.2 Environment and Application Setup

Since the application needs to be assessed for scalability, using the tool locally is not viable due to the core count limitations of VirtualBox that were mentioned previously. To address this, the environment for this test was moved to a VM in the Google Cloud Project. The Google Cloud Project compute engine was used to create a VM that runs Ubuntu 20.04.6, has 24 cores and 64 gigabytes of memory, and 200 GB of storage. This allows us to scale the number of devices in the environment up to 24.

The configuration files needed to be modified from the previous test to set up the environment for this test. Firstly, the `templates/Vagrantfile.template` was modified as follows:

```
Vagrant.configure("2") do |config|
config.vm.define "bootNode" do |bootNode|
  bootNode.vm.box = "iotBox"
  bootNode.vm.network "private_network",
    ip: "10.1.0.2",
    netmask: "255.255.255.0",
    virtualbox__intnet: "private_net"
  bootNode.vm.hostname = "bootNode"
  bootNode.vm.provision :shell, path: "scripts/setupBootNode.sh",
   env: {"IPADDRESS" => "10.1.0.2"}
  bootNode.vm.provider "virtualbox" do |vb|
    vb.name = "bootNode"
    vb.cpus = 1
    vb.memory = 1024
  end
end
config.vm.define "sensor" do |sensor|
  sensor.vm.box = "iotBox"
  sensor.vm.network "private_network",
    ip: "10.1.0.3",
    netmask: "255.255.255.0",
    virtualbox__intnet: "private_net"
  sensor.vm.hostname = "sensor"
  sensor.vm.provision :shell, path: "scripts/setupDevice.sh",  env: {
   "IPADDRESS" => "10.1.0.3"}
  sensor.vm.provider "virtualbox" do |vb|
    vb.name = "sensor"
    vb.cpus = 1
    vb.memory = 1024
    vb.customize ["modifyvm", :id, "--cpuexecutioncap", "20"]
  end
end
  #Start
end
```

Listing 15. The modified `templates/Vagrantfile.template`

This template adds the sensor, which will host the assessed application. The added sensor has the same parameters as the base devices added to the environment and runs the same setup script as mentioned previously, which sets up the private Ethereum chain used in this test.

Configuration files regarding the sensor were manually created. The wallet address and key were manually created and stored in the files `secrets/addresses/sensor` and `secrets/keys/sensor`. This was done since the sensor is static in these tests and does not require changes. Then, the rest of the configurations regarding keys and addresses were set up using the helper script `secrets/genSecrets.sh` provided in section 6.2.

6.3.3 Testing Methods

The application scenario to be tested is a situation where the sensor has to serve many vehicles. This could be, for example, a traffic jam. In this scenario, a single IoT sensor is responsible for replying to the requests of many vehicles trying to create a request for the sensor. To simplify the testing scenario, the WiFi communication between the sensor and vehicles is simulated as a stable connection, which means that issues with WiFi in this scenario do not exist.

The vehicles in this testing scenario are simulated. The vehicles run the following simulated testing client:

```
import requests
import sys
import time

walletAddr = sys.argv[1]
hostname = sys.argv[2]
```

```python
print (walletAddr)
print(hostname)
url = 'http://10.1.0.3:8080'
data = {'address': walletAddr}
started = 0
def post():
        x = requests.post(url, json = data)
        print(x.text)
while started == 0:
        try:
                post()
                started = 1
        except:
                continue
times=[]
for i in range(10):
        start = time.time()
        post()
        end = time.time()
        times.append(end-start)
path = "/vagrant/apps/" + hostname
print(path)
outputFile = open(path,'w')
outputFile.write(str(times))
outputFile.close()
```

Listing 16. The simulated vehicle client

The time it takes for a vehicle to complete a request is measured to assess how the application in the sensor scales. This is done by measuring the time to complete the request inside the simulated vehicle clients. Then this request is repeated ten times by each simulated vehicle client. These requests are made simultaneously to the sensor to estimate the impact of increased requests on the request times. The simulated vehicle client application creates requests to the sensor until it receives a valid response. At this point, the countdown of timed requests begins. This test is then run with the following amount of simulated vehicles: 3, 9, 6, and 12.

To assist in running the vehicle clients, a helper shell script `runVehicles.py` was created. The contents of the script are listed below:

```bash
num=1
for i in $( seq 1 $num )
do
   wallet=`printf "0x%s" "$(cat secrets/addresses/device$i)"`
   vagrant ssh device$i -c "tmux new-session -d -s test_session \"
    python3 /vagrant/apps/SimulatedVehicleClient.py $wallet device$i\"
    "
done
```

Listing 17. The helper script `runVehicles.py`

The variable num is used to specify how many vehicle clients are tested. The command that runs the simulated vehicle client utilizes tmux, to run the applications in the background. Once this shell script has finished, the sensor application can be started, and thus the test is started.

The expectation in this testing case is that the average response time should increase as the number of clients increases since the sensor should get overloaded with requests, thus extending the response time or failing the response.

6.3.4 Results

The results of the test are presented in Table 7. These results are not as expected. The average time to complete a request should increase, or requests should be dropped as the sensor gets increased amounts of requests. However, these results show that something is causing the time of completed requests to decline after a certain point. The tests were rerun, and while the tests were ongoing, the command `sudo top` was run on the host VM inside the Google Cloud. This revealed that the conducted tests used 100% of the available CPU power inside the VM. This, in turn, resulted in VirtualBox freezing the execution of some devices, which in turn resulted in them not properly measuring the time it took to get the answer for the request.

Vehicle device count	**Amount of failed requests**	**Average time to complete a request(s)**
3	0/30	2.896209081
6	0/60	6.102432267
9	0/90	6.008555471
12	0/120	4.815656743

Table 7. Results of validation test 2

Since these results can not be used to assess the application’s scalability, we modified the simulated vehicle client to address the issue of resource usage by adding a 1-second delay after each request using the `time.sleep(1)` in the simulated vehicle client. After the initial test, it was noticed that not adding a delay between each request would be unreasonable since, in the real-world scenario, the vehicles would not create requests immediately after receiving a response from the sensor. Instead, they would have a delay consisting of the rediscovery of the sensor. Additionally, the number of generated vehicle clients did not utilize all 24 cores of the VM since only 12 were generated. To fix these issues, the tests were rerun with the modified vehicle application and with more devices: 1,5,10,15, and 20. The results for rerunning the tests are presented in Table 8: The results of rerunning the tests presented values closer to our expectations. To visualize these

Vehicle device count	Amount of failed requests	Average time to complete a request(s)
1	0/10	1.001340389
5	0/50	5.135305734
10	0/100	10.07572883
15	2/150	10.79422091
20	3/200	14.82393002

Table 8. Results of retesting validation test 2

results, we plotted the rerun results, which is presented in Figure 13. In this plot, the x-axis contains the number of vehicle clients that send requests to the sensor. The y-axis presents the meantime of a successful request. The data points are plotted as red dots. A linear dashed line connects these data points and thus estimates the average creation times of other devices.

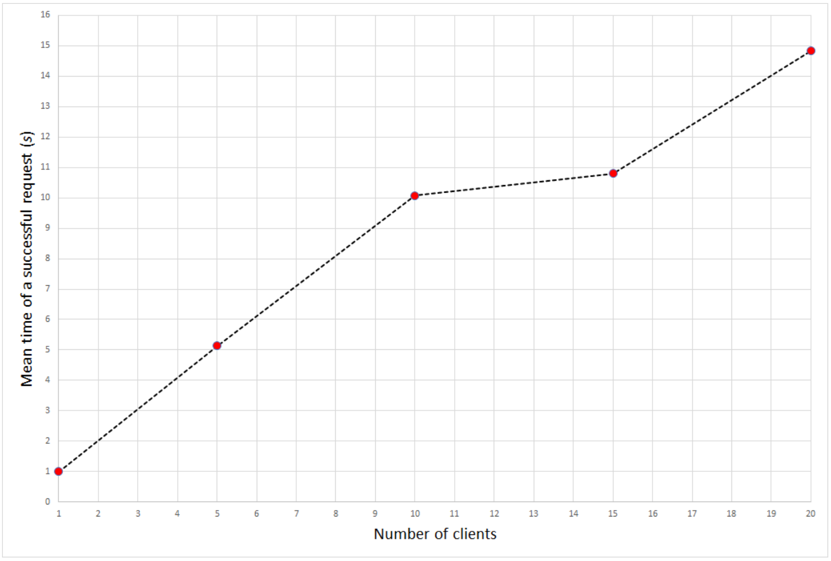

Figure 13. Plot of retesting validation test 2.

The results of the retest seem to match the expectations: As the amount of requests created by the vehicles increases, the mean time of requests increases, and when the request time is too long, requests start to fail. However, the request time should first increase linearly until requests start to fail, at which point the meantime of a request should start to settle since failed requests are not counted. The plot shows an anomaly in the data between 10 and 15 devices, where the request time is nearly identical for 10 and 15 devices.

The tested application's scalability seems limited based on the results of the tests. At 15 devices, the application cannot respond to all requests, which could be considered a failure, meaning that the application is not scalable. Additionally, at the start, the response time seems to increase linearly with the number of requests the application receives, which means the application is not scalable. This could be fixed by assigning more threads to the sensor to run the application since it serves requests using only one thread.

6.4 Summary

This chapter presented the methods that were used to validate the proposed tool. The validation tests that were conducted were presented. This included two tests: the first test targeted validating the basic functionality of using the tool by running a custom Ethereum blockchain using the provided tool. The second test then tested the scalability of a blockchain-enabled IoT application in the IoV domain. These tests had common variables which were discussed. The results of the first test were as expected. However, the second test failed the first time, which resulted in retesting. The retest matched expectations and concluded that the tested application was not scalable.

7 Discussion

This chapter addresses **RQ**$_3$ by discussing our validation methods. Section 7.1 discusses the feasibility of our scalability testing tool. Section 7.2 discusses the possible threats to the validity of our tool, such as environment creation times, and presents possible solutions to these threats.

7.1 Evaluating the feasibility of the tool

We provided a method of evaluating the feasibility of a scalability testing tool for blockchain-enabled IoT applications. This method used two validation tests; The first validation test was used to verify that the tool's basic functionality exists. Then, a scalability test was conducted, where the scalability of a blockchain-enabled IoT application was tested by measuring the mean time of application requests in contrast to increased clients.

The results of the first test matched expectations. The proposed tool was proven to generate virtual testing environments where blockchain-enabled IoT applications can be run. The tool functioned as expected in these tests. However, The second test revealed a major flaw in the environment where the test was run. The problem with the second test was that the processor capacity was maxed, which means that the test results were unreliable. To address this issue, the vehicle client application was modified to have a small delay between requests. Retesting with the modified vehicle client resulted in expected results. However, from the results, it is visible that the tool should be tested with more devices. The tool was limited to 24 virtual devices since a 24-core VM was used, which limits the maximum device count in this scenario to 24 devices (of which a maximum of 20 + 2 were used). The Google Cloud computing engine offers more powerful VMs, which could be used to run the tests. However, these VMs are unavailable for users unless the users specifically request a VM with more processing power. Additionally, the retest of the application resulted in data that had an anomaly. This anomaly raises the question of whether the data from the tests be trusted since the first run also had false data. Due to these reasons, we conclude that the proposed tool is feasible to assess the scalability of blockchain-enabled IoT devices, but only up to a maximum of 24 devices. However, the tool could be feasible to use with more devices if more data is collected by upgrading the host server and rerunning the tests with more devices.

7.2 Threats to Validity and Possible Solutions

The main threat to the tool's validity is the limited number of devices it can create. Even with high core counts, such as 124-core CPUs, it still limits the scalability tests to 124 devices. This could be fixed by changing the VM provider from VirtualBox to another VM provider, which supports assigning a single core to many VMs.

The limited amount of devices is not the only threat. Currently, the environment's startup time linearly increases as the number of devices increases. This is due to the way that Vagrant provisions machines consecutively. This can be an issue if this tool is deployed to a CI pipeline for testing or if the tests are conducted on a large scale since the tool creation time scales linearly with the number of devices created. One possible solution to improve the startup time would be to run vagrant with the option `-parallel`. However, VirtualBox does not support this command, and thus, it was not used in the environment. To solve this issue, another Vagrant provider, such as the previously mentioned Amazon EC2 cloud provider, could create these devices parallelly directly to the cloud.

One of the gaps mentioned in the previously implemented testbeds was that many utilized a physical testing environment to test their proposed blockchain-enabled IoT applications. The cost of these physical testing environments was one of the factors mentioned in this thesis as a motivator to virtualize the tool. However, running these tests in the cloud is not free since Google and other cloud providers charge for their services. The total cost for the second validation test was around 34€ (including debugging and development). This is noticeably cheaper than purchasing 20 separate devices and setting them up manually. However, running the tests in the cloud could be more expensive than building a physical testing network if these tests are repeated. For example, if the tool is used inside a CI pipeline, it could be run multiple times, which may be more expensive than investing in the hardware that runs the tests. Thus the tool should be utilized for scalability assesments.

8 Conclusion

This work explored the current scalability issues that blockchain-enabled IoT applications face. First, we conducted a SLR to assess what scalability issues exist and what tools the current literature provides to assess scalability. Based on the literature review results, we proposed a method to assess the scalability of blockchain-enabled IoT applications. From this method, it was noticed that there exists a need for a scalability testing tool. We then proposed a method for designing, implementing, and validating a tool that could be used in assessing the scalability of blockchain-enabled IoT applications. We then developed the tool for assessing the scalability of blockchain-enabled IoT applications. The tool was built using various technologies, such as Vagrant, VirtualBox, Python, and shell scripts. The tool was then tested for validity by running two separate tests. The first test assessed the validity of the tool's basic functionality. In the first test, the tool created a simulated blockchain network running on IoT devices multiple times. The second test examined the validity of using the tool for assessing the scalability of blockchain-enabled IoT applications. This was done by creating a POC application, which was then scalability tested using the provided tool. Then we examined the results of both tests and estimated the feasibility of using the tool to assess scalability. We concluded that the proposed tool is feasible to assess the scalability of blockchain-enabled IoT devices, but only in situations where the device count is a maximum of 24. This is due to the limitations in the tool and the conducted tests.

8.1 Research Question Answers

In this section, we reintroduce our research questions and present the answers to these research questions. The main research question of this thesis was *How to conduct a scalability assessment for blockchain-enabled IoT applications*. This research question is important since blockchain-enabled IoT applications have seen use in many different areas, and the area is not widely researched. Our main research question resulted in three sub-research questions, which were:

1. **RQ**$_1$ What are the primary challenges and limitations in testing the scalability of blockchain-enabled IoT applications?
2. **RQ**$_2$ How to develop a scalability testing tool for blockchain-enabled IoT applications?
3. **RQ**$_3$ How to evaluate the feasibility of the scalability testing tool?

We addressed **RQ**$_1$ by conducting a SLR. Chapter 3 provided an answer to this research question. We answered this question by first looking at the scalability issues of blockchain-enabled IoT applications. The main scalability issues were hardware, consensus, and throughput limitations. We then examined the current status of testing methods and, from them, examined the primary challenges and limitations in testing the scalability of blockchain-enabled IoT applications.

We observed that the primary challenges and limitations are the following. Firstly, there is little literature regarding the matter. Secondly, no current tools exist for generating testing environments for assessing the scalability of blockchain-enabled IoT applications.

RQ$_2$ was addressed by Chapters 2, 4 and 5. This question was first addressed in 3 by researching previous implementations in the literature. However, none were found, and thus we answered this question by designing and implementing a scalability testing tool.

Our proposal for this question is as follows. First, the tool must be designed. This includes the identification of stakeholders, modeling the goals of those stakeholders, and then creating specifications and requirements for the tool. Secondly, the tool needs to be implemented. This includes the selection of underlying technologies and implementation of the tool.

Finally **RQ**$_3$ was addressed by by Chapters 6 and 7. These chapters provided tests to evaluate the feasibility of using the tool to assess scalability. Then the results were evaluated for the tests.

We proposed the following method for validating scalability assessment tools for this question. Firstly, validation tests need to be conducted on the test. We chose to run two validation tests, one for validating the basic functionality and one for testing a POC application's scalability. Secondly, the results of the tests need to be evaluated. With our tool, we evaluated that it is unsuitable for scalability testing over 24 devices due to the tool's limitations and the tests' results.

8.2 Future Works

This section discusses the possible future works of this thesis Due to the time limitations of the thesis, many optimizations to the tool were not implemented. These optimizations are a possible direction for future works. Additionally, only one type of blockchain technology was used with the tool, which creates an opportunity to research the use of other blockchain technologies on the tool. The tool could also be utilized in other areas regarding blockchain-enabled IoT applications, such as security and privacy, which were not in the scope of this thesis. This is further discussed as a potential future work.

8.2.1 Upgrading the device count of the tool

The validation tests only created 22 devices (20 vehicles, one sensor, and one bootstrapping node). To address this issue, one possible future research direction would be to rerun the test with more devices. One option would be to use a host that provides additional CPU cores. Then, the tests could be conducted for more devices, allowing more accurate feasibility estimation for the provided tool.

Another possibility to address this issue would be to switch the VM provider of Vagrant, which is currently VirtualBox. VirtualBox has many limitations, which could be mitigated by changing the provider. One possible solution would be using a Cloud provider to create the IoT devices. For example, the Amazon EC2 platform is supported as a provider via a Vagrant plugin [8]. This would allow the creation of IoT devices in the cloud and, as a result, would allow larger-scale tests to be completed and provide faster environment creation times since devices could be created in parallel.

8.2.2 Using Different Blockchain

This thesis used only a private Ethereum chain in the tests of the tool. However, other blockchains exist, which could be a possible future work direction from this thesis. Additionally, these tests could be repeated by using another client for Ethereum, which could be a possible research direction. Another possibility would be to conduct the tests on another blockchain-enabled IoT application.

8.2.3 Tool usage in other Research Areas

The proposed tool was designed to test the scalability of blockchain-enabled IoT applications. However, the tool could be used for testing other applications as well. Security research revolving around blockchain-enabled IoT applications could utilize the tool. One possible research direction would be to deploy a malicious node in the network using the tool and then research what malicious activities the adversary could accomplish. Another research direction would be to use the tool for privacy analysis in blockchain-enabled IoT. An application could be deployed using the tool, and then the privacy of that application could be analyzed.

[8]`https://github.com/mitchellh/vagrant-aws`

References

[1] M. Nofer, P. Gomber, O. Hinz, and D. Schiereck, “Blockchain,” *Business & Information Systems Engineering*, vol. 59, pp. 183–187, Jun 2017.

[2] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” *Decentralized business review*, p. 21260, 2008.

[3] F. Yang, S. Wang, J. Li, Z. Liu, and Q. Sun, “An overview of internet of vehicles,” *China Communications*, vol. 11, no. 10, pp. 1–15, 2014.

[4] H. Atlam, A. Alenezi, M. Alassafi, and G. Wills, “Blockchain with internet of things: Benefits, challenges, and future directions,” *International Journal of Intelligent Systems and Applications*, vol. 10, no. 6, pp. 40–48, 2018.

[5] L. Tseng, X. Yao, S. Otoum, M. Aloqaily, and Y. Jararweh, “Blockchain-based database in an iot environment: Challenges, opportunities, and analysis,” vol. 23, p. 2151–2165, sep 2020.

[6] S. Singh, A. S. M. S. Hosen, and B. Yoon, “Blockchain security attacks, challenges, and solutions for the future distributed iot network,” *IEEE Access*, vol. 9, pp. 13938–13959, 2021.

[7] S. Latif, Z. Idrees, Z. e Huma, and J. Ahmad, “Blockchain technology for the industrial internet of things: A comprehensive survey on security challenges, architectures, applications, and future research directions,” *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 11, 2021.

[8] M. A. Ferrag, L. Shu, X. Yang, A. Derhab, and L. Maglaras, “Security and privacy for green iot-based agriculture: Review, blockchain solutions, and challenges,” *IEEE Access*, vol. 8, pp. 32031–32053, 2020.

[9] P. Ratta, A. Kaur, S. Sharma, M. Shabaz, and G. Dhiman, “Application of blockchain and internet of things in healthcare and medical sector: Applications, challenges, and future perspectives,” *Journal of Food Quality*, vol. 2021, 2021.

[10] J. Sengupta, S. Ruj, and S. Das Bit, “A comprehensive survey on attacks, security issues and blockchain solutions for iot and iiot,” *J. Netw. Comput. Appl.*, vol. 149, jan 2020.

[11] P. K. Sharma, N. Kumar, and J. H. Park, “Blockchain technology toward green iot: Opportunities and challenges,” *IEEE Network*, vol. 34, no. 4, pp. 263–269, 2020.

[12] M. A. Walker, A. Dubey, A. Laszka, and D. C. Schmidt, "Platibart: A platform for transactive iot blockchain applications with repeatable testing," in *Proceedings of the 4th Workshop on Middleware and Applications for the Internet of Things*, M4IoT '17, (New York, NY, USA), p. 17–22, Association for Computing Machinery, 2017.

[13] M. D. Hill, "What is scalability?," *ACM SIGARCH Computer Architecture News*, vol. 18, no. 4, pp. 18–21, 1990.

[14] L. Duboc, D. Rosenblum, and T. Wicks, "A framework for characterization and analysis of software system scalability," in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pp. 375–384, 2007.

[15] G. Bigini, V. Freschi, and E. Lattanzi, "A review on blockchain for the internet of medical things: Definitions, challenges, applications, and vision," *Future Internet*, vol. 12, no. 12, pp. 1–16, 2020.

[16] B. Cao, Y. Li, L. Zhang, L. Zhang, S. Mumtaz, Z. Zhou, and M. Peng, "When internet of things meets blockchain: Challenges in distributed consensus," vol. 33, p. 133–139, nov 2019.

[17] W. Viriyasitavat, T. Anuphaptrirong, and D. Hoonsopon, "When blockchain meets internet of things: Characteristics, challenges, and business opportunities," *Journal of Industrial Information Integration*, vol. 15, pp. 21–28, 2019.

[18] A. Mitra, B. Bera, and A. K. Das, "Design and testbed experiments of public blockchain-based security framework for iot-enabled drone-assisted wildlife monitoring," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 1–6, 2021.

[19] X. Wang, G. Yu, X. Zha, W. Ni, R. Liu, Y. Guo, K. Zheng, and X. Niu, "Capacity of blockchain based internet-of-things: Testbed and analysis," *Internet of Things (Netherlands)*, vol. 8, 2019. cited By 21.

[20] H. Sun, S. Hua, E. Zhou, B. Pi, J. Sun, and K. Yamashita, "Using ethereum blockchain in internet of things: A solution for electric vehicle battery refueling," (Berlin, Heidelberg), p. 3–17, Springer-Verlag, 2018.

[21] S. Li, L. D. Xu, and S. Zhao, "The internet of things: a survey," *Information Systems Frontiers*, vol. 17, pp. 243–259, Apr 2015.

[22] A. Taivalsaari and T. Mikkonen, "On the development of iot systems," in *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*, pp. 13–19, 2018.

[23] A. Taivalsaari and T. Mikkonen, “A roadmap to the programmable world: Software challenges in the iot era,” *IEEE Software*, vol. 34, pp. 72–80, 01 2017.

[24] B. Bera, S. Saha, A. K. Das, N. Kumar, P. Lorenz, and M. Alazab, “Blockchain-envisioned secure data delivery and collection scheme for 5g-based iot-enabled internet of drones environment,” *IEEE Transactions on Vehicular Technology*, vol. 69, no. 8, pp. 9097–9111, 2020.

[25] I. Mistry, S. Tanwar, S. Tyagi, and N. Kumar, “Blockchain for 5g-enabled iot for industrial automation: A systematic review, solutions, and challenges,” *Mechanical Systems and Signal Processing*, vol. 135, p. 106382, 2020.

[26] B. Bera, S. Saha, A. K. Das, and A. V. Vasilakos, “Designing blockchain-based access control protocol in iot-enabled smart-grid system,” *IEEE Internet of Things Journal*, vol. 8, no. 7, pp. 5744–5761, 2021.

[27] A. Dua, N. Kumar, and S. Bawa, “A systematic review on routing protocols for vehicular ad hoc networks,” *Veh. Commun.*, vol. 1, pp. 33–52, 2014.

[28] V. Buterin, “Ethereum white paper: A next generation smart contract & decentralized application platform,” 2013.

[29] L. Duboc, E. Letier, and D. S. Rosenblum, “Systematic elaboration of scalability requirements through goal-obstacle analysis,” *IEEE Transactions on Software Engineering*, vol. 39, no. 1, pp. 119–140, 2012.

[30] L. Lao, Z. Li, S. Hou, B. Xiao, S. Guo, and Y. Yang, “A survey of iot applications in blockchain systems: Architecture, consensus, and traffic modeling,” *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–32, 2020.

[31] A. Alrehaili, A. Namoun, and A. Tufail, “A comparative analysis of scalability issues within blockchain-based solutions in the internet of things,” *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 9, 2021.

[32] B. Koteska, E. Karafiloski, and A. Mishev, “Blockchain implementation quality challenges: a literature,” in *SQAMIA 2017: 6th workshop of software quality, analysis, monitoring, improvement, and applications*, vol. 1938, pp. 8–8, 2017.

[33] S. Zhu, S. Yang, X. Gou, Y. Xu, T. Zhang, and Y. Wan, “Survey of testing methods and testbed development concerning internet of things,” *Wireless Personal Communications*, pp. 1–30, 2022.

[34] Z. Rahman, X. Yi, I. Khalil, and A. Kelarev, “Blockchain for iot: A critical analysis concerning performance and scalability,” in *Quality, Reliability, Security*

and Robustness in Heterogeneous Systems: 17th EAI International Conference, QShine 2021, Virtual Event, November 29–30, 2021, Proceedings 17, pp. 57–74, Springer, 2021.

[35] A. I. Sanka and R. C. Cheung, "A systematic review of blockchain scalability: Issues, solutions, analysis and future research," *Journal of Network and Computer Applications*, vol. 195, p. 103232, 2021.

[36] A. Luntovskyy and L. Globa, "Performance, reliability and scalability for iot," in *2019 International Conference on Information and Digital Technologies (IDT)*, pp. 316–321, IEEE, 2019.

[37] W. T. Pereira, L. C. De Biase, G. Fedrecheski, and M. K. Zuffo, "A virtualized testbed for iot: Scalability for swarm application," in *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*, pp. 1074–1079, IEEE, 2023.

[38] T. go-ethereum Authors, *Go-ethereum documentation*.

[39] Oracle, *VirtualBox documentation*.

[40] "Introduction: Vagrant: Hashicorp developer."

[41] M. A. Uddin, A. Stranieri, I. Gondal, and V. Balasubramanian, "A survey on the adoption of blockchain in iot: challenges and solutions," *Blockchain: Research and Applications*, vol. 2, no. 2, p. 100006, 2021.

[42] Q. Zhou, H. Huang, Z. Zheng, and J. Bian, "Solutions to scalability of blockchain: A survey," *IEEE Access*, vol. 8, pp. 16440–16455, 2020.

[43] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," vol. 2, 01 2007.

[44] P. Urien, "Blockchain iot (biot): A new direction for solving internet of things security and trust issues," 2019.

[45] B. K. Mohanta, D. Jena, S. Ramasubbareddy, M. Daneshmand, and A. H. Gandomi, "Addressing security and privacy issues of iot using blockchain technology," *IEEE Internet of Things Journal*, vol. 8, no. 2, pp. 881–888, 2021.

[46] A. Dhar Dwivedi, R. Singh, K. Kaushik, R. Rao Mukkamala, and W. Alnumay, "Blockchain and artificial intelligence for 5g-enabled internet of things: Challenges, opportunities, and solutions," *Transactions on Emerging Telecommunications Technologies*, 2021.

[47] M. Torky and A. Hassanein, “Integrating blockchain and the internet of things in precision agriculture: Analysis, opportunities, and challenges,” *Computers and Electronics in Agriculture*, vol. 178, 2020.

[48] C. Ye, W. Cao, and S. Chen, “Security challenges of blockchain in internet of things: Systematic literature review,” *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 8, 2021.

[49] M. Jan, J. Cai, X.-C. Gao, F. Khan, S. Mastorakis, M. Usman, M. Alazab, and P. Watters, “Security and blockchain convergence with internet of multimedia things: Current trends, research challenges and future directions,” *Journal of Network and Computer Applications*, vol. 175, 2021.

[50] R. Huo, S. Zeng, Z. Wang, J. Shang, W. Chen, T. Huang, S. Wang, F. Yu, and Y. Liu, “A comprehensive survey on blockchain in industrial internet of things: Motivations, research progresses, and future challenges,” *IEEE Communications Surveys and Tutorials*, vol. 24, no. 1, pp. 88–122, 2022.

[51] J. Li, A. Maiti, M. Springer, and T. Gray, “Blockchain for supply chain quality management: challenges and opportunities in context of open manufacturing and industrial internet of things,” *International Journal of Computer Integrated Manufacturing*, vol. 33, no. 12, pp. 1321–1355, 2020.

[52] R. Memon, J. Li, M. Nazeer, A. Khan, and J. Ahmed, “Dualfog-iot: Additional fog layer for solving blockchain integration problem in internet of things,” *IEEE Access*, vol. 7, pp. 169073–169093, 2019.

[53] F. Chen, Z. Xiao, L. Cui, Q. Lin, J. Li, and S. Yu, “Blockchain for internet of things applications: A review and open issues,” *Journal of Network and Computer Applications*, vol. 172, 2020.

[54] M. A. Ferrag, M. Derdour, M. Mukherjee, A. Derhab, L. Maglaras, and H. Janicke, “Blockchain technologies for the internet of things: Research issues and challenges,” *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 2188–2204, 2019.

[55] B. Bhushan, C. Sahoo, P. Sinha, and A. Khamparia, “Unification of blockchain and internet of things (biot): Requirements, working model, challenges and future directions,” *Wirel. Netw.*, vol. 27, p. 55–90, jan 2021.

[56] C. Peng, C. Wu, L. Gao, J. Zhang, K.-L. Yau, and Y. Ji, “Blockchain for vehicular internet of things: Recent advances and open issues,” *Sensors (Switzerland)*, vol. 20, no. 18, pp. 1–37, 2020.

[57] A. Sultan, M. A. Mushtaq, and M. Abubakar, “Iot security issues via blockchain: A review paper,” in *Proceedings of the 2019 International Conference on Blockchain Technology*, ICBCT 2019, (New York, NY, USA), p. 60–65, Association for Computing Machinery, 2019.

[58] M. U. Hassan, M. H. Rehmani, and J. Chen, “Privacy preservation in blockchain based iot systems: Integration issues, prospects, challenges, and future research directions,” *Future Gener. Comput. Syst.*, vol. 97, p. 512–529, aug 2019.

[59] N. Fabiano, “Internet of things and blockchain: legal issues and privacy. the challenge for a privacy standard,” vol. 2018-January, pp. 727–734, 2018.

[60] M. Ali, H. Karimipour, and M. Tariq, “Integration of blockchain and federated learning for internet of things: Recent advances and future challenges,” *Computers and Security*, vol. 108, 2021.

[61] B. Dinesh, B. Kavya, D. Sivakumar, and M. R. Ahmed, “Conforming test of blockchain for 5g enabled iot,” in *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, pp. 1153–1157, 2019.

[62] J. Fat and H. Candra, “Blockchain application in internet of things for securing transaction in ethereum testnet,” vol. 1007, 2020. cited By 2.

[63] S. Eisele, I. Mardari, A. Dubey, and G. Karsai, “Riaps: Resilient information architecture platform for decentralized smart systems,” in *2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC)*, pp. 125–132, 2017.

[64] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, “Hyperledger fabric: A distributed operating system for permissioned blockchains,” in *Proceedings of the Thirteenth EuroSys Conference*, EuroSys ’18, (New York, NY, USA), Association for Computing Machinery, 2018.

[65] R. Dua, A. R. Raja, and D. Kakadia, “Virtualization vs containerization to support paas,” in *2014 IEEE International Conference on Cloud Engineering*, pp. 610–614, 2014.

Acknowledgements about the use of AI tools

This thesis utilized two different AI tools in the writing process. Most notably, the Grammarly Chrome plugin [9] developed by Grammarly Inc. (2023) was used. The Grammarly Chrome plugin is an AI tool that can check grammar, spelling, and punctuation. Grammarly was utilized in this thesis to fix grammatical errors and rephrase sentences. The Chrome plugin of the tool was used with version 14.1119.0.

The second AI tool utilized in this thesis was ChatGPT 3.5[10] (OpenAI, personal communication, 2023). ChatGPT was utilized to overcome writer's block and as a brainstorming tool. ChatGPT is a language model chatbot developed by OpenAI (2023).

[9] `https://app.grammarly.com/`

[10] `https://openai.com/blog/chatgpt`

Appendix A: Issues of Blockchain-enabled IoT Applications

Study	Issue category													
	Direct scalability issues					**Privacy & security issues**					**Social & legal issues**			
	Limited storage	**Limited processing power**	**Throughput**	**PoW inefficiency**	**Network limitations**	**Device security**	**User privacy**	**Smart Contract security**	**Combined attacks**	**Transaction privacy**	**Implementation costs**	**Lack of technical standards**	**Lack of Legal standards**	**Lack of expertise**
[7]	x							x						
[4]	x	x			x								x	x
[8]	x	x		x										
[5]	x	x												
[48]	x	x				x						x	x	
[6]	x		x			x			x					
[15]	x		x											
[47]	x				x									
[9]	x					x								
[49]		x		x		x						x	x	
[50]		x				x						x		
[16]		x									x			
[51]		x	x				x				x			
[11]			x	x							x	x	x	
[10]				x	x									
[17]			x	x										
[52]			x											
[53]			x	x			x	x		x				
[54]				x					x					
[55]				x		x	x							
[56]						x								
[60]							x			x				
[57]							x							
[58]							x							
[59]							x				x			
[46]							x						x	

Appendix B: User Guide for the Tool

This Appendix describes how the virtual testing environment tool can be installed, deployed, and modified to be used with other IoT-based applications. A quick setup guide is demonstrated in a demo video.

Github Repository

The GitHub repository of the tool is located at `https://github.com/SanttuSi/IoTSimulation`. The tool contains three branches. The first branch, the main branch, contains the base version of the tool. The second branch is called the validationTest1 branch and contains the source code used in the first validation test. The third branch is called the validationTest2 branch and contains the source code used in the second validation test.

Installing dependencies and downloading the tool

The tool uses multiple different CLI tools that are included with Ubuntu 20.04.6. However, using Ubuntu 20.04.6 is not required to use the tool but may require installing some additional CLI tools. In addition to these tools, the environment uses some programs not included in Ubuntu 20.04.6. The programs that are not included in Ubuntu 20.04.6 and are a prerequisite for the environment are VirtualBox, Vagrant, and the eth-keys and eth-hash[pycryptodome] Python module. These programs can be installed by using the following commands in a Linux bash terminal:

- `sudo apt-get install virtualbox`

-
  ```
  wget https://releases.hashicorp.com/vagrant/2.3.2/vagrant_2.3.2-1_amd64.deb
  sudo apt install ./vagrant_2.3.2-1_amd64.deb
  vagrant plugin install vagrant-vbguest
  ```
- `sudo apt install pip`
- `pip install eth-keys`

After installing the dependent programs, the environment repository can be downloaded by utilizing the `git` program:

- `git clone -b validationTest2 git@github.com:SanttuSi/IoTSimulation.git`

Setting up and deploying the environment

Once the installation is complete, the user can now generate the private keys, wallet addresses, and the genesis.json file. The user can either manually do this or the user can navigate to the folder `secrets/` and run the script:
`./generateSecrets.sh`
This script automatically generates three private Ethereum keys and 3 Ethereum wallet addresses. The keys are stored in the folder secrets/keys/, and the addresses are stored in the folder secrets/addresses/. The genesis.json file is stored in the secrets/ folder.

It is also possible to generate different amounts of keys and addresses for the environment. For example, if the `./generateSecrets.sh` script is executed with the following format `./generateSecrets.sh -n 10`, the script will generate ten keys and ten wallet addresses and add funds to those wallets in the generated genesis.json file.

Once the setup is complete, the user can generate the environment. From the project's root folder, the user can run the `./startup.sh` script, which deploys the entire environment and starts up the application on the nodes. By default, the script brings up 3 IoT devices. The number of nodes can be modified by providing the flag `-n` paired with the number of nodes, similar to the `./generateSecrets.sh` script.

Reconfiguring parameters for the environment

This environment can be used to test out other blockchain-enabled IoT applications. However, it requires modifications to some of the configuration files in the project. Many files in the environment can be configured. However, from a user's perspective, the most important configuration files allow to change the properties of nodes or the application deployed on the nodes. These configuration files are listed in Table 7. The table presents the use case of the file in the environment and what possible modifications the user can make to these files.

File name	Description	Possible modifications
`iotBox/Vagrantfile`	Used to create the base OS to be used in all devices	Using a different operating system
`iotBox/setup.sh`	Used to provision the base OS in iotBox/Vagrantfile	Installing a different application
`scripts/setupBootNode.sh`	Used to setup and run the boot node application	Setup a different boot node, setup a miner for the network
`scripts/startup.sh`	Used to setup and run the Geth application for the IoT devices	Setup and run a different application
`templates/Vagrantfile.template`	Used to configure the final Vagrantfile and used to define static nodes, such as bootnodes	Defining a miner for the network
`templates/device.template`	Used to define the hardware and networking properties of each node	Using other hardware configurations, reconfiguring the network
`templates/genesis.json.template`	Used to define the genesis block of the private Ethereum chain on the network	Using different consensus method, starting from a different block

Table 9. Configuration files, their purpose, and possible modifications in them

Appendix C: Licence

Non-exclusive license to reproduce thesis and make thesis public