Mittuniversitetet
MID SWEDEN UNIVERSITY

A Proposal and Implementation of a Novel Architecture Model for Future IoT Applications

With focus on fog computing

Viktor Andersson

Master thesis – Institution of Information Systems and Technology
Main field of study: Computer engineering
Credits: 30
Semester/year: Spring, 2022
Supervisor: Stefan Forsström, stefan.forsstrom@miun.se
Examiner: Tingting Zhang, tingting.zhang@miun.se
Programme: Master of Science in Engineering: Computer Engineering

Abstract

The number of IoT devices and their respective data is increasing for each day impacting the traditional architecture model of solely using the cloud for processing and storage in a negative way. This model may therefore need a supporting model to alleviate the different challenges for future IoT applications. Several researchers have described and presented algorithms and models with focus on distributed architecture models. The main issues with these however is that they fall short when it comes to the implementation and distribution of tasks. The former issue is that they are not implemented on actual hardware but simulated in a constrained environment. The latter issue is that they are not considering sharing a single task but to distribute a whole task. The objective of this thesis is therefore to present the different challenges regarding the traditional architecture model, investigate the research gap for the IoT and the different computing paradigms. Together with this implementing and evaluating a future architecture model capable of collaboration for the completion of a generated task on multiple off-the-shelf hardware. This model is evaluated based on task completion time, data size, and scalability. The results show that the different testbeds are capable communicating and splitting a single task to be completed on multiple devices. They further show that the testbeds containing multiple devices are performing better regarding completion time and do not suffer from noticeable scalability issues. Lastly, they show that the completion time drops remarkably for tasks that are split and distributed.

Keywords: Internet of Things, Fog computing, Cloud computing, Mist computing, Distributed Systems

Acknowledgements

I would like to thank my supervisor Stefan Forsström for always being available and supportive throughout the process of this thesis.

Table of Contents

Table of Figures

Table of Tables

Terminology

Acronyms/Abbreviations

IoT	Internet of Things
IIoT	Industrial Internet of Things
RPi	Raspberry Pi
AI	Artificial Intelligence
ML	Machine Learning
DRL	Deep Reinforcement Learning
FL	Federated Learning
AR	Augmented Reality
GIL	Global Interpreter Lock
STDEV	Standard Deviation
GPS	Global Positioning System
TCP/IP	Internet Protocol Suite
TCP	Transmission Control Protocol
IP	Internet Protocol
UDP	User Datagram Protocol
ARP	Address Resolution Protocol
ICMP	Internet Control Message Protocol
RFID	Radio-Frequency Identification
NFC	Near Field Communications
WSAN	Wireless Sensor and Actuator Networks
RWA	Random Walk Algorithm

NFA	Neighboring Fogs Algorithm
NOA	No Offloading Algorithm
OFA	Optimal Fog Algorithm
QoE	Quality of Experience
NE	Nash Equilibrium
DSRM	Design Science Research Methodology
FIFO	First-In-First-Out
CSV	Comma-Separated file
IQR	Interquartile Range method
LF	Lower Fence
UF	Upper Fence

Mathematical notation

N	Number of nodes
C_N	Number of simultaneous connections
t_T	Total completion time
t_{list}	Time for requesting and receiving list of connected nodes
$N(t_{help})$	Time to ask for help from N nodes
t_{data}	Time to send the message containing one half of the task
t_{sort}	Time it takes to sort the task
t_{result}	Time for the result to be sent back to the source node
$t_{finished}$	Timestamp when a task is completed
$t_{generated}$	Timestamp when a task is generated
Q_3	Upper Quartile

Q_1	Lower Quartile
n	Total number of data points
x	Data point
$\bar{x}$	Average completion time or data size.
s	Standard deviation
P_{list}	Number of permutations of the list of connected nodes

1 Introduction

This chapter describes background and objectives regarding the area of Internet of Things (IoT) and its architecture model.

1.1 Background and problem motivation

IoT applications has so far made a great impact on connecting the physical world with the digital world using embedded devices, sensors, and electronics. When combined with cloud computing and its computational power, the capabilities, and possibilities for IoT applications extends even further. This combination of IoT devices and cloud is today often the traditional architecture model for these types of applications. The architecture could be described as a many-to-one relationship, where multiple IoT devices are connected to a cloud server for processing and storage. This type of architecture can be seen in figure 1.

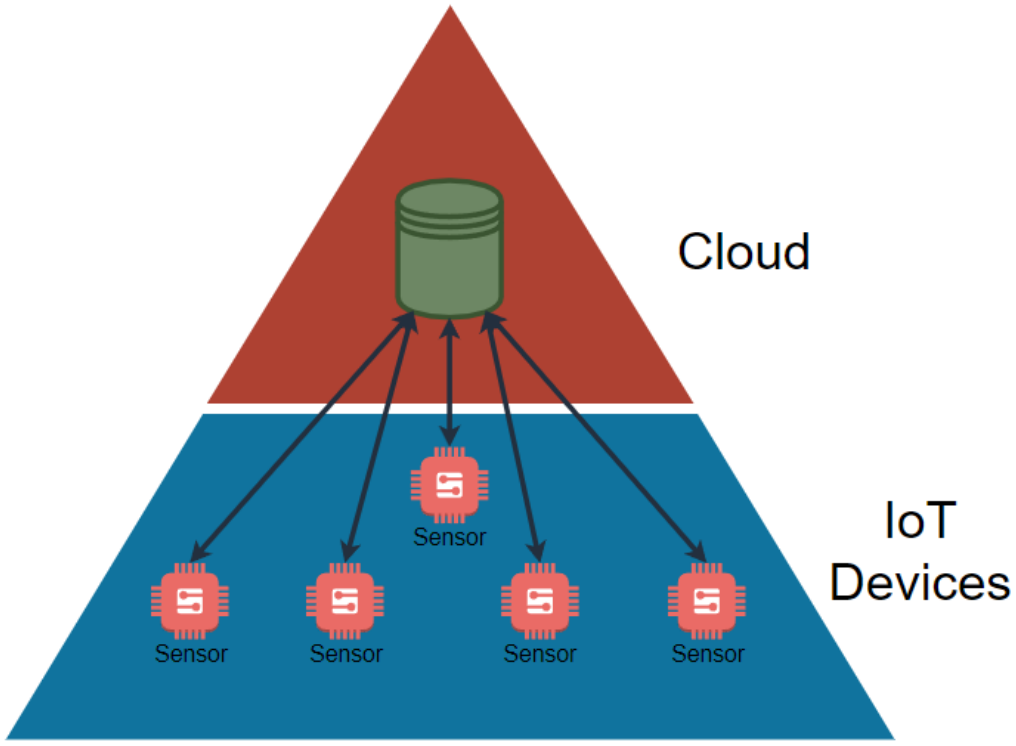

Figure 1: Traditional IoT and Cloud architecture

Even though this type of architecture provided an almost revolutionary effect in its early stages, we are now encountering its challenges more

and more. Both IoT device-related but also the sole use of the cloud for computational power and storage. The former presented by Yaqoob *et al.* [1] being interoperability, scalability, flexibility, energy efficiency, mobility management, and security. A selection of the latter presented by Sadeeq *et al.* [2] being security, reliability, and storage.

We can however see several new potential applications on the horizon, given that the supporting functions inside the systems become better. For example, better utilization of resources, and specialized adaptations based on different scenarios. This provides possibilities to utilize resources more locally situated instead of the traditional solution with external cloud computing solutions for IoT and Industrial Internet of Things (IIoT). Many of the technologies and functions embedded in IoT and IIoT are powerful in their own domain and in specific applications, unfortunately there are few efforts into combining them all into a larger testbed that enables distributed pooling of resources. A testbed that can show a potential future architecture model and back-end system for the next generation of IoT applications.

Because of this, the ambition of this work is to set up a novel architecture model. Together with this, investigate the limits and possibilities as well as show proof-of-concept applications on the system, and find future research directions.

1.2 Overall aim

The overall aim of this work is to give insight and enlighten the potential issues with the traditional architecture model for existing and future IoT applications. A complementary resource will therefore be offered to support these applications, mitigating challenges, and making the flow and operations in the systems more effective.

1.3 Problem statement

The investigated and researched problem in this thesis is the potential issues of solely using the traditional cloud-based solutions for future IoT applications. Together with this providing an optional solution in the form of a fog computing focused internet architecture testbed, which should be capable of IoT intercommunication and collaboration in a distributed manor. Such a system needs to be evaluated in terms of energy consumption, response times, cost, scheduling, reliability, scalability, security, throughput, availability, and privacy. Hence, we

need to evaluate and compare the different aspects of this problem, as well as perform physical measurements in a real-world testbed.

1.4 Research questions

From this problem statement there emerges four research questions which will be answered in this thesis. These research questions are the following:

RQ1: Why and to what extent is distributed pooling of resources needed for future IoT applications?

RQ2: How can multiple IoT-devices collaboratively work towards completing tasks?

RQ3: How effective is the proposed novel architecture model in terms of task completion time, data size, and scalability?

RQ4: What are the benefits and drawbacks of the proposed novel architecture model?

Hence, the contribution of this work will be to design, implement, run, and evaluate a novel design for a future internet architecture model for the IoT. Doing so exploring an often challenge-remarked area of IoT namely the collaboration of fog devices or nodes.

1.5 Scope

The focus of this thesis lies on presenting the different gaps of research in the area of IoT and fog computing, designing and implementing a novel architecture model, and lastly showing the benefits of this model as an option to the traditional architecture. Because of this, the development of a traditional architecture model using a cloud service will be out of scope, but a server-like computer will be used as a reference model. The implementation of the novel architecture model will have a maximum of three nodes working simultaneously in a network with an external coordinator.

1.6 Outline

Chapter 2 describes the theory for the main areas of this thesis namely IoT, Mist/Fog/Cloud-computing, and related work. Chapter 3 presents the method chosen to accomplish the goals of the thesis. Chapter 4

describes the different approaches that can be taken. Chapter 5 describes the implementation of the multiple testbeds. Chapter 6 presents the results of evaluation of the testbeds. In Chapter 7 the results, method, and ethical and societal aspects are analyzed. Chapter 8 concludes the work of the thesis where research questions are answered, and potential future work is presented.

2 Theory

This chapter will cover the main technological areas of this thesis and lastly related works. The first technological area to be presented is the IoT with the different computing paradigms surrounding it, namely, mist, fog, and cloud computing. The second technological area is distributed systems and its different branches.

2.1 Internet of Things

The main reasons for the success and continuous development of IoT are presented by Corcoran [3] to be because of cloud computing together with the technological advancements of mobile devices and data networks. The former explained as dating back to the 1960's but has up until around 10 years ago boomed into what is now seen as necessary and something that is almost taken for granted. Enabling photos, videos, account information, and all sorts of files to not occupy space on each device related to the user, but to store it in a remote accessible place for later use on any internet-enabled device. The latter with the introduction of the smart phone enabling people to walk around with a powerful handheld device small enough to fit in a pocket in a pair of pants. With this not only acting a regular telephone, but combining technologies such as calculators, video cameras, computers, Global Positioning System (GPS), and much more.

The connection between the cloud and the devices has as explained by Corcoran [3] been since its early stages been done mainly using the Internet Protocol Suite (TCP/IP). TCP/IP as described by Kale and Socolofsky [4] is generic term that symbolizes everything related to the more specific protocols Transmission Control Protocol (TCP) and Internet Protocol (IP). Multiple protocols and technologies have been layered on top of TCP/IP, such as User Datagram Protocol (UDP), Address Resolution Protocol (ARP), and Internet Control Message Protocol (ICMP). More regarding TCP is that it provides a connection-oriented byte stream which guarantees delivery of packets between devices.

The most prominent and enabling technology for the IoT is described by Atzori *et al.* [5] as Radio-Frequency Identification (RFID) where they also mention Near Field Communications (NFC), and Wireless Sensor and Actuator Networks (WSAN). Liu and Lu [6] presents these technologies

and mentions more such as infrared sensors, GPS, internet and mobile networks, network services and industry applications. However, they proceed to explain that RFID and WSAN are the two most important and foundation building technologies for the IoT.

The global expansion and progress of IoT and its connected devices are estimated by [7] to be 14.4 billion globally. With this said it has become apparent that the IoT is here to stay and will continue to grow emerging into more and more applications. Kaur and Singh [8] present five present and future application domains to be the following: Transportation and Logistics; Healthcare; Smart Environments; Personal and Social; Futuristic Applications. The domains and corresponding areas are presented in table 1.

Table 1: Summary of IoT domains and their corresponding areas [8]

Domain	**Application area**
Transportation and logistics domain	Logistics Assisted driving Mobile ticketing Monitoring environmental parameters Augmented maps
Healthcare domain	Tracking Identification and authentication Data collection Sensing
Smart environments domain	Comfortable homes and offices Industrial plants Smart museum and gym
Personal and social domain	Social networking Historical queries Losses Thefts

Futuristic applications domain	Robot taxi City information model Enhanced game room

2.2 IoT – Cloud, fog, mist

In the survey done by Nord *et al.* [9] it is stated that there is no consensus of a standard definition of IoT, however they do present a general idea of what the IoT is, "a network of networks of uniquely identifiable end points or "things" that capture and share data". Continuing they also describe multiple different architecture models proposed by different sources with the common pattern of consisting of at least three layers. The objective of each layer in the sources they investigate differs slightly, but they all have the common structure of having three layers. The layers are namely a sensing layer, a network layer, and an application layer. The sensing layer being responsible of sensing surroundings and generating data, the network layer transforming and processing the generated data, and lastly the application layer that finalizes the processing and presents it in context relevant for the wanted system or solution. The architecture model is presented in figure 2.

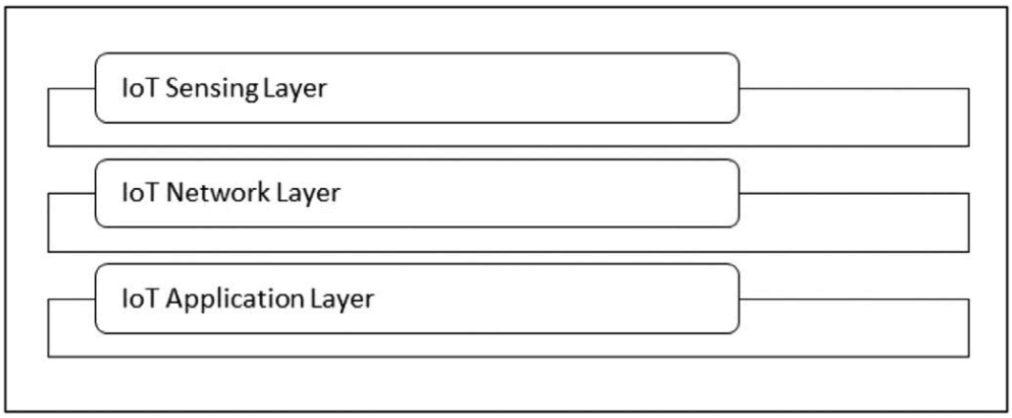

Figure 2: IoT architecture model [9]

The breakthrough for IoT as described by Bellavista *et al.* [10] was a combination of the smart phone together with the cloud computing. The former connecting every owner of a smart phone to the internet and the latter enabling heavy workloads generated in the smartphone to be processed on a powerful, often non-local server, popularly called the

cloud. Due to this the network layer has been cloud focused, much because of the ability to store and process data remotely without putting a large stress on the IoT device. This was until recent years since the processing power of IoT devices has increased rapidly and different drawbacks and challenges has started to appear with the traditional focus of the cloud.

Pan and McElhannon [11] investigate and discusses the future for IoT applications in relation to cloud computing. They present three benefits and three challenges with the traditional cloud computing model. The benefits being the on-demand payment method, flexible and scalable resources, and lastly provides great processing capabilities that for example supports Machine Learning (ML). The challenges they describe are the following: the increase of IoT devices which impacts the volume and velocity of data that these devices send up to the cloud; latency caused by the potential large distance from the IoT devices to the datacenters; a monopoly on the infrastructure that cloud computing possess since only giant companies can afford those resources, causing others to be dependent on these types of services. They propose the concept of open edge cloud infrastructure which moves the cloud closer to the IoT devices where they investigate multiple state-of-the-art research efforts and technologies in this area. The summary of related efforts that they provide can be seen viewed in appendix A.

From this, emerged two other computing paradigm, fog computing and mist computing which have been included into the IoT network layer.

Bellavista *et al.* [10] describes fog computing as a distributed computing paradigm with focus on moving previous cloud responsibilities closer to the IoT devices, negating the possible issues with cloud computing such as delay, and network overload. The idea behind fog computing however is not to replace another computing paradigm but to function as a middle-tier resource for work that previously had only been done in the top-tier, the cloud. Fog computing can therefore function as a gateway where data can be processed and/or be passed on to the cloud.

There are multiple present and future application areas where fog computing could be useful, three of these are described by Yi *et al.* [12] as being the following: Augmented Reality (AR) where virtual objects can be presented in real world settings with a quicker response, increasing

user experience; Content Delivery and Caching by optimizing and utilizing user experience based on near proximity data; lastly Mobile Big Data Analytics where latency issues of the cloud could be negated by distributing workload previously only sent to the cloud.

Regarding mist computing, the bottom-tier, Vasconcelos *et al.* [13] and Galambos [14] both differentiates fog and mist computing partly by the devices involved. To further understand the difference of the two paradigms, Galambos [14] also produces the definition of mist computing covering components such as sensors and actuators with the capabilities of small preprocessing capabilities.

A novel architecture model with focus on the IoT network layer and the distribution of work contained inside, can from the definitions be presented as the figure 3.

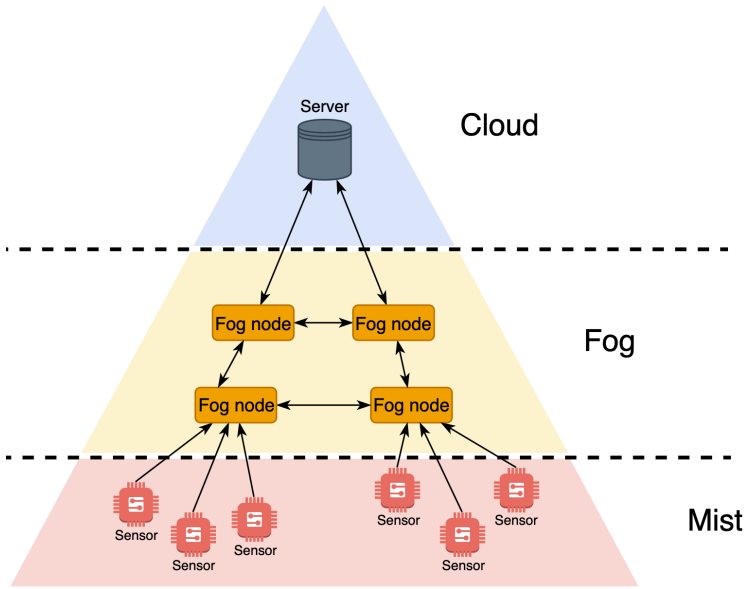

Figure 3: Novel architecture model

2.3 Distributed systems

According to Van Steen and Tanenbaum [15], there are multiple descriptions and definitions of what distributed systems are where they conclude that none of these are grasping the essence of distributed systems. They then provide the loose characterization of describing it as ”a collection of autonomous computing elements that appears to its users as a single coherent system”. They proceed to break down their

characterization in two features. Feature one being that components of a distributed system needs to be able to independently operate, where each component is referred to as a node. Feature two being that the viewer or user perceives the system as a single entity.

Van Steen and Tanenbaum [15] also presents the different types of distributed systems, namely distributed computing systems, distributed information systems, and pervasive systems. The proceeding theory of this subchapter will focus on distributed computing systems more specifically high-performance distributed computing. For this type there are two subtypes, cluster computing and grid computing, where cluster computing will be the focus of this thesis. Cluster computing refers to the usage of multiple closely connected nodes operating with homogeneous properties. While grid computing on the other hand separates the properties of each node creating heterogeneous nodes.

2.4 Related work

The subchapters that follow will present a total of three related works with focus on offloading strategies for fog and cloud computing.

2.4.1 Improving Fog Computing Performance via Fog-2-Fog Collaboration

Al-Khafajiy *et al.* [16] proposes a fog-to-fog collaboration scheme that manages offloading of incoming tasks to the fog network. By doing so formulating a mathematical model for the offloading scheme, performing experiments, and lastly evaluating their model in comparison to other offloading schemes.

Their proposed load balancing scheme is described to have two approaches. A centralized approach which relies on a master node for the distribution of workload, and a distributed model where each node in the fog network will periodically update all nodes about their workload. Even though they mention two approaches they adopt the distributed model with the motivation of being more suitable for applications where devices might be in a moving state e.g., mobile phones. The offloading occurs when a fog node that receives a request or workload is too busy with other requests. The decision for the offloading is based on the expected time for the new request to be finished. While the fog node is trying to calculate the expected time for the task to be computed on itself, the fog node sends a request to all other fog nodes

for their potential collaboration of the task. In short, the task is sent or not sent to another fog node based on the expected time for it to be completed and if no fog node has a reasonable expected time for finishing the task it is sent to the cloud.

Based on latency of different packets and their different sizes, they state that the collaboration scheme is simulated and evaluated against two other schemes, Random Walk (RWA), and Neighboring Fogs (NFA). The results showed that the proposed scheme outperforms the two others when measuring the average latency up until 7 fog nodes. When increasing the number of fog nodes above 7 however, the results show that the proposed model performs worse than the two others. Figure 4 shows the results from the study, where they refer to another algorithm called No Offloading (NOA), and their proposed scheme as Optimal Fog (OFA).

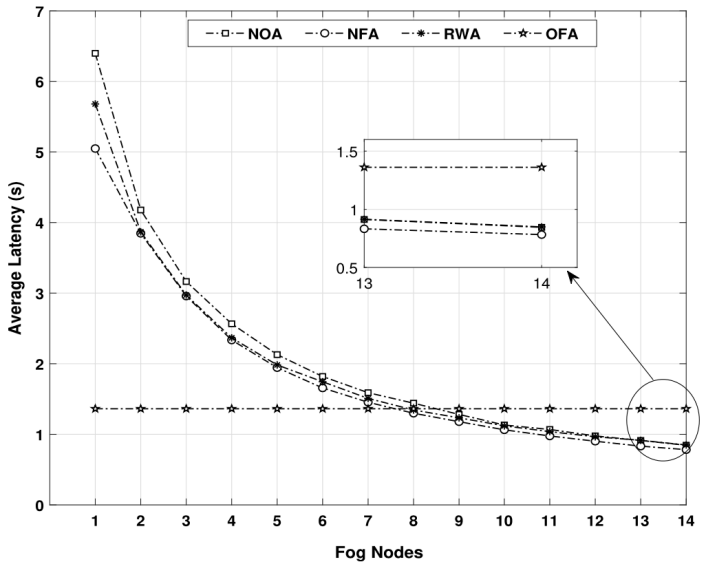

Figure 4: Result of performance for multiple algorithms [16]

They conclude that offloading models affect the latency in a significant positive manor and that they are obligatory for the potential success of fog computing.

There are mainly three differences regarding the paper in comparison to this thesis, the focus on mobile fog nodes, decentralized and distributed ways of communication, and scalability. This thesis focuses on non-mobile fog nodes. The decentralized and distributed way of

communication goes hand in hand with the scalability aspects which the paper does not bring up. The way the paper describes the communication of fog nodes is by broadcasting requests to all fog nodes in the same network asking for collaboration. While in this thesis a fog node asks for collaboration to one node at a time, not risking scalability issues that are prone to emerge when the number of fog nodes increases. The broadcasting is not explained as a one-way communication but rather a two-way communication where the broadcaster expects an answer back from every fog node it has broadcasted a message to.

2.4.2 Hierarchical fog-cloud computing for IoT systems: a computation offloading game

Shah-Mansouri and Wong [17] studies fog computing resource management with the viewpoint of the user. They propose an offloading scheme based on the idea of gamification trying to maximize the quality of experience for users by allocating processes on fog nodes in an efficient way.

Since the system is of hierarchical structure it is made up of cloud servers, fog nodes, and what they call IoT users which generates tasks of a predetermined load. The devices in their system are made up of virtual machines acting as fog nodes and cloud servers each having a dedicated set processing power. The structure they propose can be seen in figure 5.

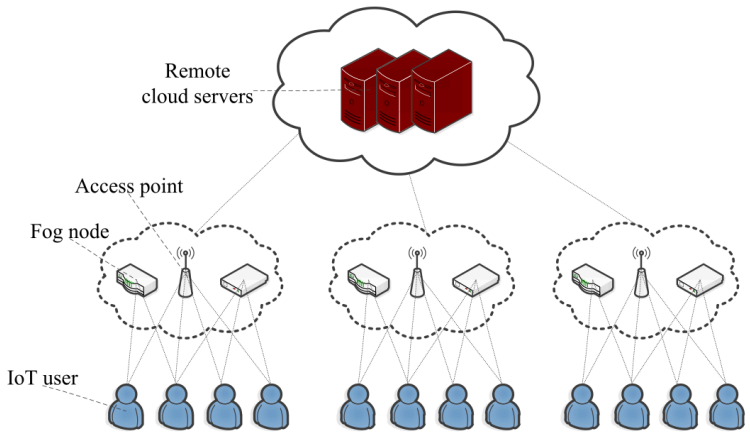

Figure 5: Structure of fog-cloud computing [17]

The IoT users have the ability to either process data by themselves or by offloading a generated task to a fog node or worst case, a remote cloud server. The decision of offloading is based on a problem they name as

Quality of Experience (QoE). This problem computes the energy and delay for the scenarios of not offloading and offloading, where the variables are then compared based on each scenario. The gamification idea stems from having all users trying to optimize the utilization of resources not by selfishly exploiting the resources for its own benefits but for the collective quality of experience of all users. They refer to this as pure Nash Equilibrium (NE), which described by Maskin [18] as ”a stationary point of an iterative adjustment process”.

The results from the study are divided into three areas, average QoE, average delay, computation time, and number of beneficial users. The results for the QoE shows that it increases drastically when increasing the number of fog nodes from 0 to 20, and then continues to increase but in a slower rate. Figure 6 shows this relationship.

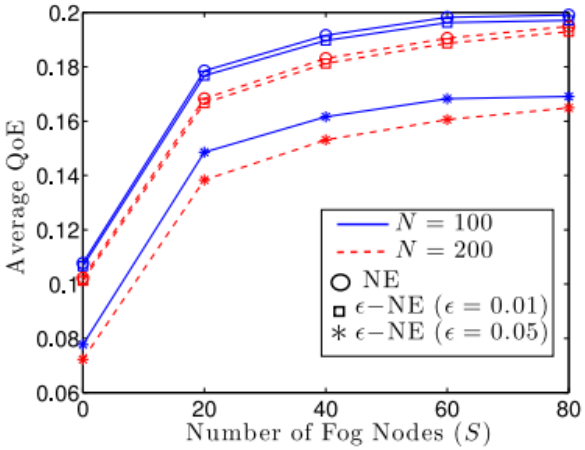

Figure 6: Results of QoE [17]

The result for the average delay shows that the computation time lowers when the number of fog nodes increases. However, the communication time increases as a consequence of this but the total average delay for the network seems to decrease regardless. Figure 7 presents this result.

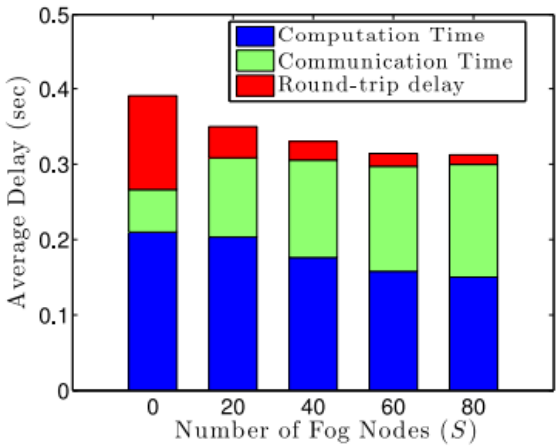

Figure 7: Result of average delay [17]

The result from the final evaluation area, number of beneficial users, show that it has an increasing effect parallel with the increase of fog nodes but has a relatively small increase for the interval 40-80 number of fog nodes. Figure 8 visualizes these results.

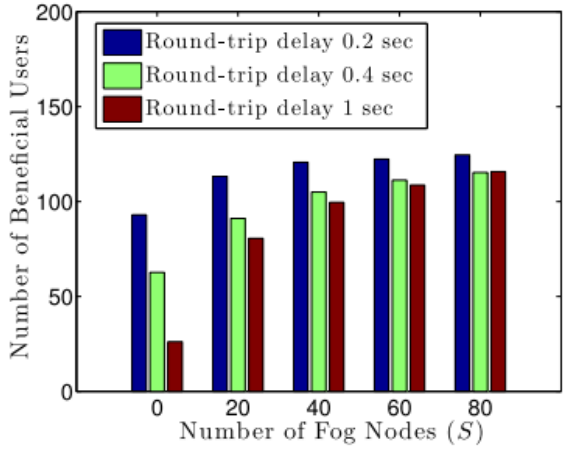

Figure 8: Result for number of beneficial users [17]

The authors conclude that the fog nodes help in providing effective computing services for users together with increasing users experience when using the proposed scheme in comparison to other related works.

In contrast to this thesis, the paper does not split up the generated task to multiple fog nodes but sends it as a complete task to one available fog node. This thesis also does not focus on a quality of experience as a collective but as every node seeing to their own needs.

2.4.3 Computation Offloading with Multiple Agents in Edge-Computing-Supported IoT

Shen *et al*. [19] studies computation offloading optimization for the IoT and from this proposes a novel offloading algorithm based on Deep Reinforcement Learning (DRL) and Federated Learning (FL). The paper uses the term edge computing when in this thesis that term is split into fog and mist computing. For the section of the paper, the term edge computing will be used instead. DRL is described as positively boosting multiple outcomes such as energy consumption and latency. They do however conclude that today's IoT devices are not powerful enough to train a large neural network. To alleviate this issue, they propose FL as a way of bypassing the training burden of each node. They divide their contribution into three aspects. The first being the issues of optimization for computation offloading, the second being design of a novel offloading algorithm with focus on federated learning, and lastly evaluating the two previous aspects by simulating the proposed algorithm.

The optimization of the offloading is done by selecting random nodes to perform training where different weights are updated accordingly. After each training session the weights are updated for every node in the network. From this optimization the network can make its decision of offloading a task or not to a node based on the previous training done by other nodes. Tasks generated for every node have a set data size but with a set probability of being generated, namely 0.1, 0.5, and 0.9.

The results from their simulation show that when the probability of a task being generated is set to 0.9 the network will have a hard time managing the task. When set to 0.5 the workload is heavy but not unmanageable, and when set to 0.1 the network is well manageable and stable. These results are shown in figure 9.

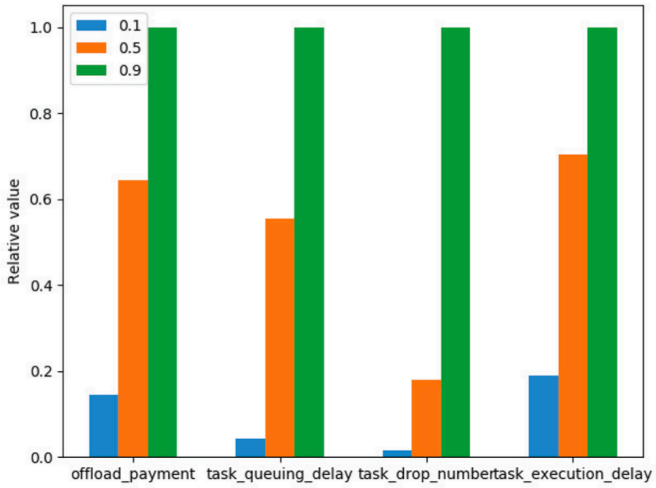

Figure 9: Result of relationship between task generation and a stable system [19]

For the completion of a task which they call task execution delay, they show that when comparing the FL-based training to two other training methods, namely centralized and greedy, the FL-based training is not performing as well. Figure 10 shows these results.

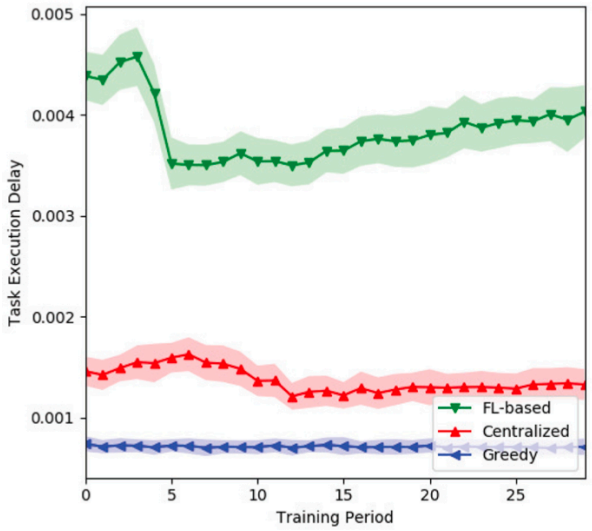

Figure 10: Result of task execution delay [19]

They conclude that FL-based DRL is an applicable and enabling technique for optimizing offloading for IoT devices. Giving the IoT devices the ability to make their own decisions.

The paper differs from this thesis by applying machine learning on IoT devices, together with embracing a maximum queue size for each node and when reached throws newly created tasks. The paper does not introduce the property of splitting a task but keeps them as one entity and offloading them in a case when its needed.

3 Methodology

This chapter will provide information regarding methods that will be used to answer the research questions. Followed by that will be the project method describing in more detail how the overall work will be conducted throughout the timeline of this thesis. Lastly the method chosen for evaluation of the thesis will be explained.

3.1 Scientific method description

The method chosen for the overall work of the thesis is the process model Design Science Research Methodology (DSRM) proposed by Peffers *et al.* [20]. This method is defined as a framework consisting of six phases, Identify Problem and Motivate (Phase one), Define Objectives of a Solution (Phase two), Design and Development (Phase three), Demonstration (Phase four), Evaluation (Phase five), Communication (Phase six). These steps are to be appropriately used for the project and function as supporting guidelines in the process of acquiring knowledge of a problem, working towards a solution and in the end having proposed a solution for the problem. Because of this, the method will be conducted in a quantitative manor. The extent of each phase is varied depending on where the focus lies on the specified research problem. A more in-depth description of the different phases will be described in subchapter 3.2 Project method description. An overview of DSRM is presented in figure 11.

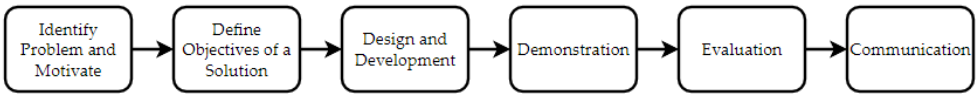

Figure 11: Simplified DSRM Process Model

The first research question (RQ1) regarding why and to what extent distributed pooling of resources is needed for the future IoT applications will be explored and answered in phase one and two of DSRM. This is done by investigating the timeline of IoT applications through literature studies exploring past, present, and possible future applications. These studies include both challenges and advantages of today's traditional architecture model.

The second research question (RQ2) exploring how multiple IoT devices collaboratively can work towards completing a task will be answered in phase three and four of DSRM. This network architecture will contain of-the-shelf hardware acting as fog nodes where each node will generate tasks and compute it or distribute parts of the task to other fog nodes in the network. The novel architecture model will focus on the fog layer.

The third research question (RQ3) regarding the effectiveness of the stated future architecture model compared to the traditional architecture model will be answered in phase five of DSRM. The different measurements that will be evaluated are completion time for tasks, data size, and scalability.

The fourth research question (RQ4) exploring the benefits and drawbacks of the proposed future architecture model will be answered in phase five and six of DSRM. This will be done by comparing it with different aspects of the traditional architecture model suggesting improvement areas for both architectures and discuss where they can support each other.

3.2 Project method description

Phase one will lay the foundation for the coming work, because of this it is of great importance to first conceptualize the problem to then be able to provide a solution for it. Therefore, this phase will be broken down into four milestones. The first milestone (M1.1) being a literature study covering at least five related works covering IoT-applications, Cloud computing, Fog computing, Mist computing, and the issues surrounding primarily cloud computing. The second milestone (M1.2) is to present at least three challenges of the traditional architecture model it potentially may encounter in the future. The third milestone (M1.3) of phase one will be to define four research questions and the contribution of this thesis.

Phase two further extends the objectives of phase one. In this phase the objectives for the solution will be defined. This is achieved by four milestones. The first milestone (M2.1) of this phase is to define the scope of the thesis, stating what will be the main focus. The second milestone (M2.2) is to define the milestones for each phase, making it clear whether that phase has been completed or not in the end. The third milestone (M2.3) is to define a specific task for the testbed to compute together with

defining the evaluation areas such as scalability, task completion time, data size.

Phase three involves the creation of an artifact. The artifact in this context is defined as a future architecture model for upcoming IoT applications. More specifically, this phase aims to develop the desired functionality and design of the future architecture model. The phase is to be divided into four milestones, where the first three are aimed at designing and developing the Mist layer (M3.1), Fog layer (M3.2), and Cloud (Reference) layer (M3.3). The last milestone (M3.4) is to combine the different layers into a collaborative intercommunicative network.

Phase four will be to demonstrate that the developed testbeds from phase three works as intended. This will be done by running the testbeds and present the findings for the supervisor of this thesis.

Phase five will be to evaluate the developed solution from M3.4 based on the defined evaluation areas from M2.3. Because of this, the evaluation method throughout this thesis will be performed in a quantitative manor.

Phase six is the last phase where the findings of the project will be communicated. This will be done through a written technical report presenting the following: an *introduction* to the area of the thesis; *theory* regarding the technology used; *method/methods* used throughout this thesis; different *approaches*; description of the *implementation* of the novel architecture model; *results* from the measured evaluation areas; a *discussion* regarding the results and chosen method; *conclusions* explaining if the research questions has been answered and what future work can be done to complement the work of this thesis. Together with the technical report, a presentation at the end of the period of the thesis work will be performed.

3.3 Evaluation method

The evaluation method will be divided into the six different parts/phases from the project method description. Each specific milestone will not be the target for evaluation but each phase as a whole. Each phase will be evaluated by answering the following questions:

- What, if any, were the challenges of the phase?

- What, if any, decisions could have been made different and how would that impact the phase?
- Could the phase be improved by reiterating the phase?

The findings of the evaluation and the answers provided for each question will be provided in the chapter 7 and 8.

4 Approach

This chapter will cover four areas which are discussed, presented, and finally chosen. The first area being the identified research gap of IoT and fog computing. The second one being the option of computational task where the N-queen problem, sorting algorithms, and image processing are presented. The third one presents the different options of communication for the testbed. The fourth area is presenting the testbeds different configurations, such as variable values and hardware specifications.

4.1 Identified research gap

Multiple works have concluded that there is a gap in the research regarding IoT, fog computing and their relationship. This gap needs to be filled in order to support future applications for when cloud computing eventually is not enough. Some of these gaps and different challenges are presented in this chapter.

Varshney and Simmhan [21] states six challenges as their version of gaps in the research, that the area of fog computing needs to address in order to make it feasible for future applications. These are the following: *Programmability* - making it easy for developers to run applications on the fog, defined as; *Predicting demand* - predicting users demands and place resources where it is needed for these demands; *Power and Network Consumption* – providing enough power and energy for each different application accordingly; *Where to push the tasks* – deciding where the tasks are to be computed for the best interest of the application; *Security and fault tolerance* – ensuring the safety for the data and users together with reliability and system safety nets; *Fog providers and billing* – how to economically charge users making it beneficial for both users and providers. As a complement to these challenges, they also do a reality check where they discuss that it is still the early days for fog computing. They explain that a threshold for the lack of fog computing has not yet been met but that the issues are emerging more and more in specifically integrated private scenarios.

The analysis and survey performed by Mahmud *et al*. [22] concluded nine possible future directions and gaps that needs to be further researched. These are the following: *Context aware resource/service provisioning* – which leads to efficient and relevant resource allocation; *Sustainable and*

reliable fog computing – optimizes economic and environmental influence; *Interoperable architecture of fog nodes* – aims to utilizing a fog nodes capabilities and not only using them as gateways or networking components; *Distributed application deployment* – further extends the previous area but aims more at the use of distributed architectures and using all resources available; *Power management with fog* – relates to how to properly handle large amounts of requests and how power consumption increases by increasing the fog; *Multi-tenant support in fog resources* – relates to how to properly handle multiple users on one single fog resource; *Pricing, billing in fog computing* – how to address the cost of fog networks and how it differentiates from cloud computing; *Tools for fog simulation* – real world testbed development or simulators for evaluating fog computing architectures and algorithms; *Programming languages and standards for fog* – due to structure differences in comparison to cloud computing, standards and programming languages need to be extended to support fog applications.

Mahmud *et al*. [23] discusses and proposes seven future research directions for the area of IoT and fog computing. They present them to be the following: *Trade-off between energy and accuracy* – having a reasonable and appropriate relation between energy and accuracy based on the application; *Artificial intelligence-based application management* – how to use artificial intelligence (AI) for managing resource allocation and potential network failures; *Pricing and detailed estimation of Fog resources* – how to appropriately charge for the use of fog resources; *Trusted service orchestration in Fog* – how to handle security for the fog depending on private or public infrastructure; *Fog node consolidation and scaling* – when and how to apply appropriate resources when needed; *Application-specific management* – policy development for specific IoT scenarios; *Task sharing and re-usability* – optimizing computational load on fog nodes by sharing tasks.

What can be concluded from these three works is that they share and agree on what research gaps need to be filled for the potential success of fog computing. The gaps that this thesis aims to investigate and contribute to are those previously presented as *Fog node consolidation and scaling, Task sharing and re-usability*, *Interoperable architecture of fog nodes, Distributed application deployment*.

4.2 Computational task

There exists three commonly used computational tasks in the area of computer science that are of interest for this project. The different computation tasks presented here and their corresponding algorithms that solve them are categorized as divide and conquer techniques. These tasks are the N-queen problem, sorting algorithms, and lastly image processing.

The N-queen problem is a chess related puzzle problem where N number of queens are placed on an N-by-N board not being able to be overtaken by another queen Kesri *et al*. [24]. A queen in chess can move in three ways, diagonally, by row, or by column, hence requiring the placement of a new queen to not be in the line of being overtaken by another queen. Figure 12 provides an overview of how a solution can look like for a 4-queen problem, i.e., the placement of 4 queens on a 4-by-4 board.

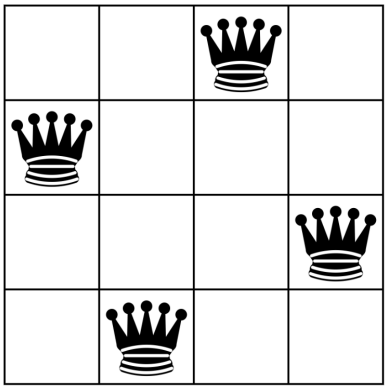

Figure 12: Solution to a N-queen (N=4) problem

A sorting task done by an algorithm comes with multiple different choices since there are multiple sorting algorithms, better or worse to choose from. The sorting algorithm requires an array of some sort either with one or multiple dimensions. The benefits of this task are the relative ease of implementing it, combined with being stable and scalable without becoming unmanageable when tuning different parameters for the computations. Another benefit is that it can be implemented on devices with everything from small to large processing capabilities. Several sorting algorithms are described and compared by Prajapati *et al*. [25], presenting relevant properties of each algorithm. Due for the need of a divide and conquer algorithm together with having an average case of good time complexity only two are selected as candidates from their

research. These two algorithms with their corresponding properties can be viewed in table 2.

Table 2: Properties of two sorting algorithms

Algorithm	**Time complexity**			**Advantage**	**Disadvantage**
	Worst case	**Average case**	**Best case**		
Merge sort	$O(N * logN)$	$O(N * logN)$	$O(N * logN)$	Efficient for small amount of input data	Requires additional memory
Quick sort	$O(N^2)$	$O(N * logN)$	$O(N * logN)$	Efficient for large amounts of input data	Inefficient for sorted data

The task of image processing is also a reasonable candidate for the systems computation part. More specifically linear neighborhood filtering (convolution), where a filter is applied to an image producing a distorted image as a result. The practical process of convolution is performed by moving pixel by pixel through an image and for every pixel applying a matrix of set size to the current pixel and its neighbors. Figure 13 shows an example of this process where *f(x,y)* is the original image, *h(x,y)* is the filter, and *g(x,y)* is the resulting image.

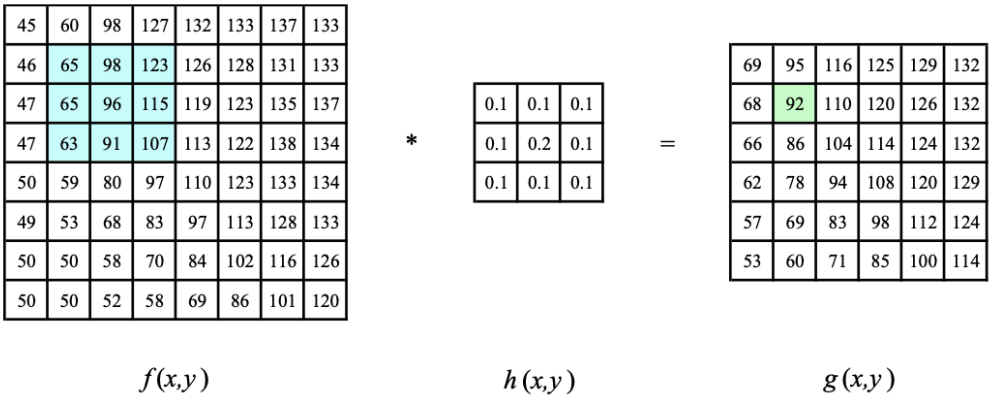

Figure 13: Image convolution [26]

This task has several beneficial properties similar to a sorting algorithm, these being the ease of implementation together with requiring one or multiple dimensional arrays. As with the sorting algorithm, there exists multiple different filters, but the computational complexity remains mainly the same for linear neighborhood filtering.

The computational task chosen for the testbeds is a sorting algorithm. The reason for this is the ability to finely tune different parameters, such as the range of numbers and the total size together. In comparison to the N-queen problem and image processing where the if the size is incremented one step, it will increase exponentially i.e., making it hard to finely tune when testing and evaluating. The sorting algorithm chosen is the merge sort algorithm, even though it requires additional memory, and the quick sort algorithm is more efficient for large amounts of input data. The reasoning behind this is a combination of quick sort having a time complexity worse than that of merge sort in the worst-case scenario, together with merge sort requiring additional memory. Requiring additional memory is not considered a disadvantage for the evaluation criteria of this thesis.

4.3 Communication orchestration

There are mainly two options of communication orchestration that are potentially viable for the task of the system. The two different options can in turn be divided into the usage of an internal coordinator or an external coordinator orchestration. The coordinator has in both options the responsibility of distributing information about the other nodes in the network making sure a newly connected node has opportunity to connect to every other node.

The first option operates by maintaining all connections to and from a node to throughout each node's existence. This is done by either having a partly external coordinator node only managing newly connected nodes and the distribution of a list of all connected nodes to that node or by assigning one of the nodes with the responsibilities of a coordinator node together with being having the workload of a regular node. A benefit to this type of connection is that the step of creating a new connection is only needed when adding a new node to the. A drawback to this type of connection is the possible scalability issues when

increasing the number of nodes in a network. This is because the total number of simultaneous connections C_N from one node to the other nodes in a network of N nodes will increase greatly as described by the formula 1.

$$C_N = N - 1 \quad (1)$$

An overview of these two orchestrations can be seen respectively in figure 14.

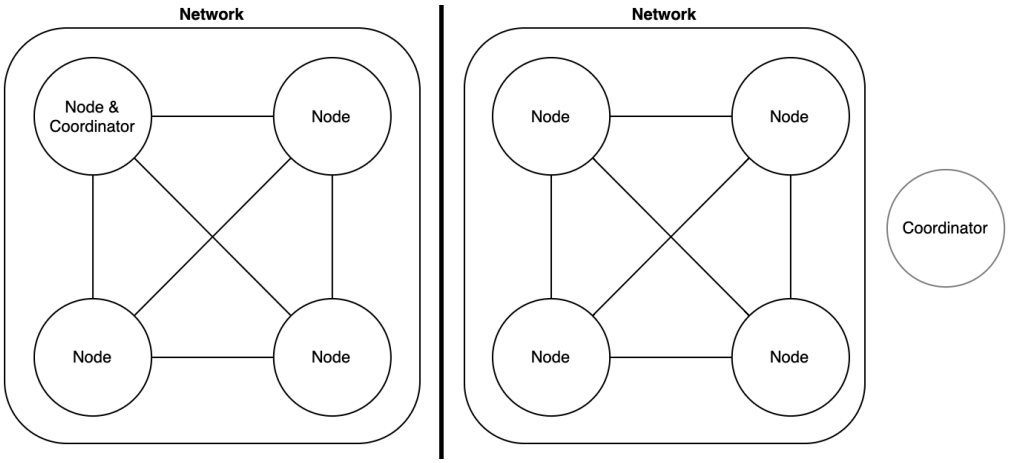

Figure 14: First option of communication orchestration, Internal versus External coordinator

The second option operates in contrast to option one by only maintaining a connection between nodes when help is needed for the computation of a task. This option can also use either an internal node with the responsibility of coordinator and regular node or an external coordinator. The benefit of this solution is the property of scalability since a connection between two nodes is broken after a shared task has been completed. The difference can be viewed in figure 15.

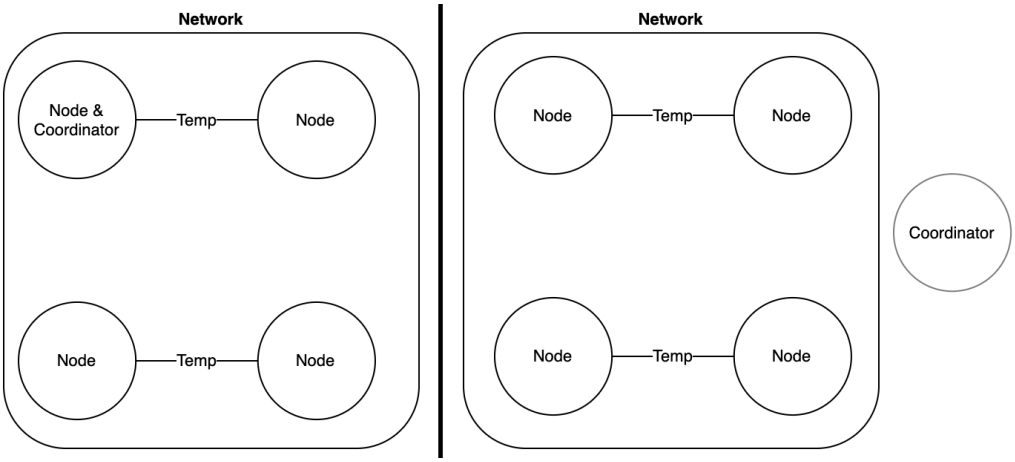

Figure 15: Second option of communication orchestration, Internal versus External coordinator

The chosen communication orchestration is the second option where an external coordinator is used. The motivation for this is the property of scalability together with separating a node's job with the coordinators. The separation provides potential greater capabilities for the nodes due to one node not needing to do the work of the coordinators. The first option provided in this subchapter is from a scalability perspective not viable but could in a very small network of nodes be reasonable. However, the scalability property is of great importance for the novel architecture model proposed in this thesis, making option two the chosen communication orchestration.

4.4 Testbed configuration

There are multiple variables that can be considered for the configuration of the testbeds, together with different settings for these as well. These variables partly impact each other which leads to careful consideration when tuning these for the system not to be overloaded or underloaded. The *generation time* describes the interval of when data is to be generated in seconds. The *data size* describes the interval of how many elements there are to be present in an array which is referred to as a task. The *threshold for needing help* refers to when a fog node needs help with a task, where the options for it can be to observe queue size, data size, RAM usage, CPU usage, or combinations of these. The *threshold for helping others* refers to when a fog node will be able to help another fog node with a task, where the options for this can be to observe queue size, help

queue size, RAM usage, CPU usage or a combination of these. The different variables and their configuration options can be seen in table 3.

Table 3: Variable configurations

Configuration variables and their options		
Generation time (s)	**Min value**	0 to 10
	Max value	0 to 100
Data size (elements in array)	**Min value**	0 to 10^3
	Max value	10^3 to 10^6
Threshold for needing help	**Queue size**	0 to 100
	Data size (elements in array)	Min to max of data size
	RAM (%)	0 to 100
	CPU (%)	0 to 100
Threshold for helping others	**Queue size**	0 to 100
	Help queue size	0 to 100
	RAM (%)	0 to 100
	CPU (%)	0 to 100

The situation of an overloaded system is partly affected by the hardware which the testbed runs on, this is because different hardware has different properties and specifications. Table 4 presents the specifications for the top tier nodes that are considered for the system.

Table 4: Top and middle tier node specifications

Top/middle tier nodes			
Type	**Specifications**		
	OS	**CPU**	**RAM**
PC	Windows 10 (64-bit)	3.80GHz Intel Core i5-7600K	16 GB
MacBook Pro	MacOS Monterey Version 12.3.1	1,4 GHz Quad-Core Intel Core i5	8 GB

Table 5 presents the specifications for the middle/bottom tier nodes to be used, the motivation behind middle/bottom is because the sensor values from a potential bottom tier node are to be simulated on the middle tier for this work.

Table 5: Middle and bottom tier node specifications

Middle/bottom tier nodes			
Generation of RPi	**Specifications**		
	OS	**CPU**	**RAM**
4th Gen. model B [27]	Raspberry Pi (32-bit)	Quad core 1.5GHz Broadcom BCM2711 64-bit	4 GB

The motivation behind the final chosen values of the variables comes from testing the different options presented in table 3. Keeping the generation time at the maximum values for a small data size will keep the system stable but will not demand or need the assistance of another node, making the communication orchestration redundant, which is not desired. On the other hand, generating the maximum sized data within

the smallest time frame will overload the system potentially making it crash due to the hardware limitations. This led to the generation time being set to range from 0 to 10 and the data size for a generated task to range from 10^4 to 10^5. The two thresholds chosen is also based on testing different values. The threshold for a node needing help is based on a combination of two configurations, queue size and the data size. Where the threshold is reached when either the queue size is 5 or more, or when the data size for the generated task exceeds the median of the maximum and minimum value of the possible data size. The reasoning behind these values is that the communication shall not be performed for all tasks, but for tasks that are of greater computational load, and also when a fog node is getting overwhelmed by tasks generated and added to its queue. The values for this threshold will not change for the different testbeds. The threshold for helping others is based on the condition that the queue size and help queue size will need to be lower than that of a specific value, where they will change for the different testbeds. The reasoning behind these values is that a fog node should not help others with tasks if it can't manage its own generated tasks or the tasks it has previously offered to help others with. This threshold for the different testbeds is therefore set to the queue size and help queue size to not exceed 0, 1, 2, and 4 respectively. The final chosen values are presented in table 6.

Table 6: Chosen configuration variables

Configuration variables and their set values		
Generation time (s)	**Min value**	0
	Max value	10
Data size (elements in array)	**Min value**	10^4
	Max value	10^5
Threshold for needing help	**Queue size**	>= 5
	Data size (elements in array)	$> \frac{10^5 + 10^4}{2}$

Threshold for helping others	**Queue size**	< (0, 1, 2, 4)
	Help queue size	< (0, 1, 2, 4)

5 Implementation

This chapter presents the technologies and the steps taken to construct the four following testbeds: reference, one-node, two-node, three-node. The general concept and overview for a fog testbed can be viewed in figure 16.

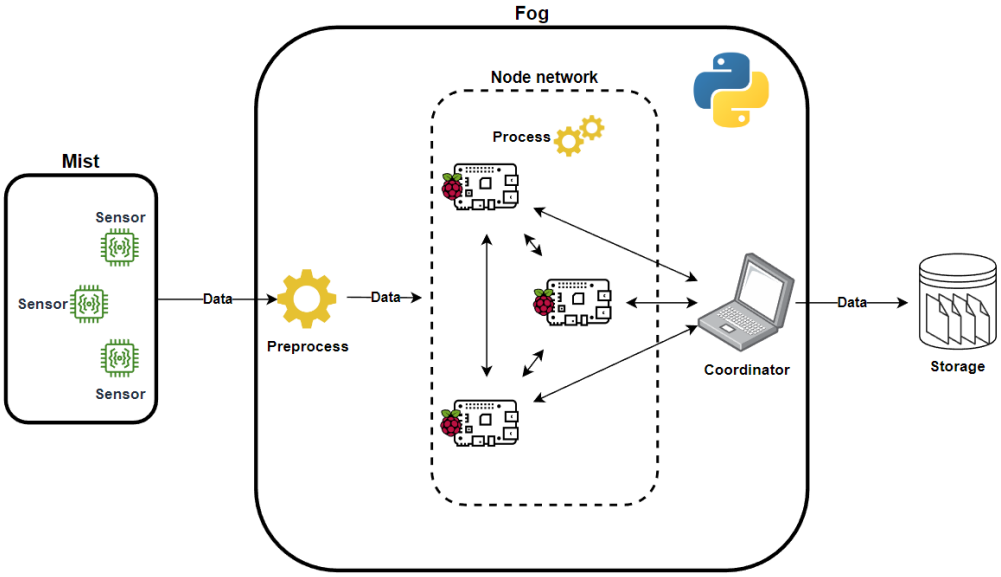

Figure 16: Fog testbed

5.1 General implementation

The implementation is performed by using the programming language Python also referred to as CPython [28] which is the traditional or standard implementation of what is regularly referred to as Python. There are however multiple branches of CPython namely: *IronPython* – which runs on .NET; *Jython* – which runs on Java Virtual Machine; *PyPy* – which uses a JIT compiler; *Stackless Python* – Which supports microthreads; *MicroPython* – which runs on micro controllers.

The use of CPython impacts the implementation of this thesis by having the property of the Global Interpreter Lock (GIL). The GIL [29] is a mechanism which makes sure that only on thread can execute at a time. This has the effect of ensuring the safety of not encountering concurrent access to a resource. This does however impact the parallelism negatively when regarding efficiency.

Most of the components presented below are operating on separate threads, which is done by using the library *threading* [30]. Due to the use of CPython as mentioned previously this library does not have the properties of a true multi-threaded program but is stated by [30] to be an appropriate model for running multiple I/O-bound tasks simultaneously. The components which are not running on a separate thread are *Notify coordinator*, and the sub-component *Send help request*. The motivation behind this for the former is because it is a gateway-requirement for the proceeding work of the other components, and for the latter being that it runs on the *Compute* component.

All connections made in the implementation of the testbeds are made by using the library *socket* which is a low-level networking interface used for connecting devices to each other. The standard protocol and the protocol used for this thesis are TCP/IP.

An overview of the different components and their relationship to each other can be viewed in figure 17 and will be presented accordingly in the subchapters that follow.

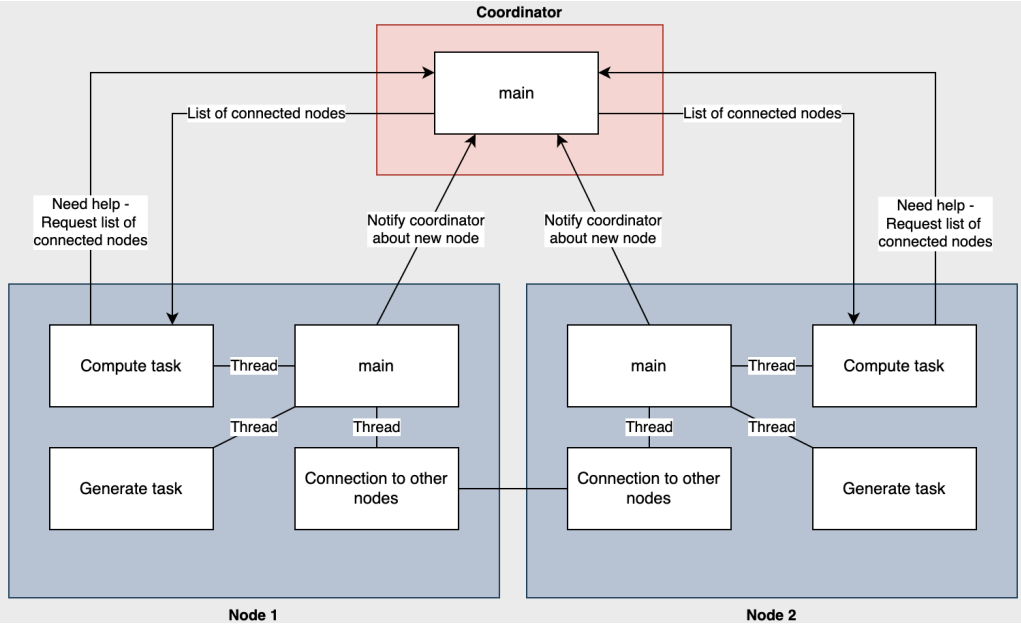

Figure 17: Testbed components

5.2 Notify coordinator

The first step for a node entering the network is to connect and notify the coordinator of the network. The coordinator saves the IP-address of this

node and adds it to a list of all existing nodes which it later will send to a node that requests help. The list is a JSON-object and the structure of it is presented in figure 18.

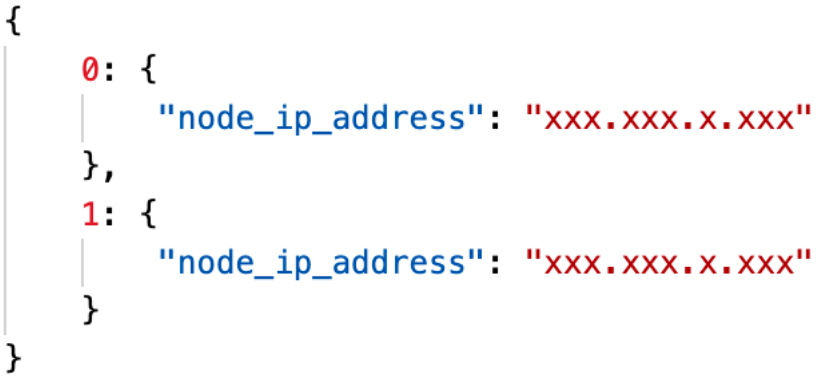

Figure 18: JSON-object of the list of connected nodes

When the coordinator has added the new node to the list, the connection is closed, and the proceeding components can start to operate.

5.3 Generation of task

The generation of data is performed by randomly generating unsorted arrays of random size at random intervals. This generation of data is what is also defined as a *task* in this thesis.

The generation of data is performed with the help of a routine from the library NumPy called Random Sampling [31]. This routine generates random numbers based on the 128-bit implementation of O'Neill's permutation congruential generator [32] which by the library is named as PCG64. The random numbers generated are used for the creation of the unsorted arrays specifying the content of the arrays, the size of the arrays, and the time for how long this thread is should sleep before generating the next array. The variables used for generating the random values is presented in table 6.

The data is processed by the same thread after creation by inserting it into JSON-objects which is then placed into a queue for further processing and computation. Before this, the size of the queue and the data is observed to determine if the node needs help or not. The thresholds for this are presented in table 6.

The data and relevant attributes of each JSON-object are the following: *data* in the form of a random sized array containing random integers; *node_address* keeps track of the IP-address of the node that has generated the data; *part_one_sorted* helps in keeping track if one part of the data is sorted; *part_two_sorted* helps in keeping track if the second part of the data is sorted; *alone* states if there are no other nodes to help; *compute* a node if it should perform the computation for the data; *help_needed* informs if a node needs help; *time_src* keeps track of the time for a task used for the evaluation; *sent* helps in the evaluation informing if another node has helped with a task. The task and its structure can be seen in figure 19.

```
{
    "data": [0,3,2,1],
    "node_address": "xxx.xxx.x.xxx",
    "part_one_sorted": true/false,
    "part_two_sorted": true/false,
    "alone": true/false,
    "compute": true/false,
    "help_needed": true/false,
    "time_src": 0,
    "sent": true/false
}
```

Figure 19: A task object and its attributes

The attribute *time_src* is set by using the library *time*, which includes multiple time-related functions [33]. The function used in this implementation is *perf_counter()*, which returns the float value of a clock with the highest available resolution, i.e., it acts as a stopwatch.

5.4 Communication

This component can be divided into two sub-components that define the communication between the nodes. The first one is where a node needs help and sends the request to another node. The second one being where a node receives a request from another node wanting help with a task. These two sub-components are presented below.

5.4.1 Send help request

This sub-component operates by connecting to the coordinator to fetch the list of connected nodes. The list is randomized with the *shuffle*-function from NumPy [34]. This list is then used to connect to every node in the list. With each connection, the node needing help sends a byte encoded message indicating that it needs help. The node then waits for a response on whether it gets help or not. Based on the response and a scenario where the node that needs help is alone in the network, the component will change three attributes of the task. These are the attributes *alone*, *compute*, and *help_needed*.

If the list only contains the node needing help, it will change the attributes in the mentioned order to *true*, *true*, and *false*.

If help is not granted it closes that connection and continues through the list until either help is granted, or the list has been processed. If the list has been processed and no other node will help, it will change the attributes in the mentioned order to *true*, *true*, and *false*.

If help is granted, it first changes the attributes of the task in the mentioned order to *false*, *true*, and *false*. Secondly it sends the task as a JSON-object to that helping node. To minimize overhead of the transmitted data, the task is filtered by removing different attributes not necessary for the other node to complete the task. This object includes the following attributes: the first *data* is an identifier for the helping node that it is data or task that is being received; the *task_id* is an index for the task keeping track of the task; the second *data* tells the helping node what task is to be computed; *node_address* is the source nodes' IP-address; *part_two_sorted* is an indicator for if the second part of the task has been completed. This JSON-object can be seen in figure 20.

```
{
    "data": {
        task_id:{
            "data": [2,1],
            "node_address": "xxx.xxx.x.xxx",
            "part_two_sorted": false,
        }
    }
}
```

Figure 20: JSON-object of a *data* message

5.4.2 Receive help request

This sub-component acts as a server, waiting for connect requests from other nodes needing help. This connection then accepts encoded messages, decodes them, and checks what type or identity of message it is. The message identity can be one of the following:

- *help* – indicating that the connecting node needs help, where the potential helper responds with a *yes* or a *no*. If *yes*, the helper waits for the next message, if *no*, closes the connection. The answer depends on the thresholds presented in chapter 4.4.

- *data* – indicating that the message is a JSON-object that contains data to be computed by the helper, which the helper adds to a separate queue dedicated for data that has come from other nodes. This queue is referred to the help queue. An example of this message can be seen in figure 20. The connection is closed after the task has been added to the computation queue of the helper and will be re-opened by the helper when the task has been completed according to the *ifixed* message.

- *ifixed* – indicating that the message is a JSON-object that contains the result and appropriate attributes from a helping node that has finished its part of a task. These attributes excluding *ifixed* are the following: *task_id* – stating the ID generated for the task at the source node; *data* – which is the result of the computation performed by the helping node; *part_two_sorted* – which states that

the task has been completed. This JSON-object including the message identifier *ifixed* can be seen in figure 21.

```
{
    "ifixed": {
            "task_id": task_id,
            "data": [1,2],
            "part_two_sorted": true,
    }
}
```

Figure 21: JSON-object of an *ifixed* message

- No message – closes the connection.

Large messages that are sent through the connections are divided into multiple messages due to the nature of the *socket* library and its function *recv()*. The messages received on the helping side needs therefore to be read as fragments. The fragments can at the end be concatenated producing the original message in full. An example of this can be seen in figure 22.

```
fragment = ''
while true:
    receive fragment
    is fragment null:
        break
    fragment = fragment + received fragment
    if end of message:
        break
```

Figure 22: Pseudo-code for reading fragments of a message

5.5 Compute

This component performs the computations on the task residing in the compute queue and the help queue where tasks are processed according in a first-in-first-out (FIFO) order for both queues. The priority is however the compute queue. The computation is done by using non-library merge sort algorithm. The computation depends on the attributes previously mentioned in chapter 5.3.1, for if help has been granted or not.

If help is not granted for the task, the complete task is computed by the node that generated the task (source node). If help is granted, the source node will compute the first half of the task and then set the attribute *part_one_sorted* to *true*. The node will then try to process the first task residing in the help queue if there is any. If a task exists, it will process that task accordingly and send the task with relevant attributes back to the source node of that task and lastly remove that task from the help queue. When finished with the first task or no task exists in the help queue, it will check the attribute *part_two_sorted* of the first task in the compute queue again to see if the whole task has been completed and can be removed from the compute queue. Figure 23 shows the flow of this component.

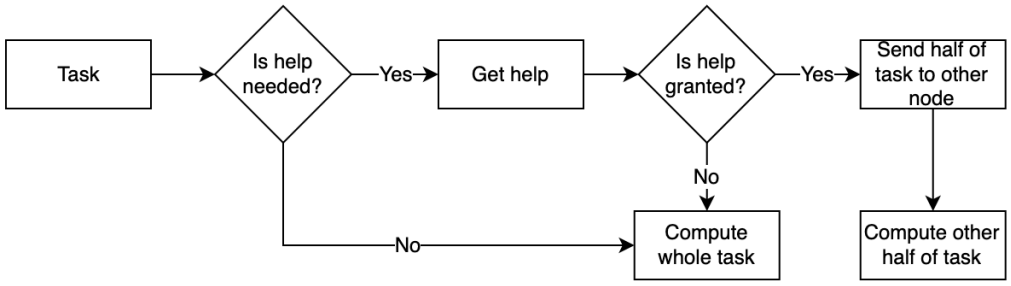

Figure 23: Flow of the compute component

5.6 Measurement setup

In accordance with the evaluation areas the different measurements taken are completion time for a task, data size, and scalability. Completion time for a task, and data size are together with the id for a task, and a variable stating if the task has been split and sent or not are sent to a Comma-Separated file (CSV) for further analysis in Excel. A simple example of this file can be viewed in figure 24.

ID	Sent	Size	Completion time
0	True	15000	1600
1	False	12000	2000

Figure 24: Example of the CSV file for the different measurements

The measurements are performed the same for all testbeds and corresponding thresholds.

The completion time for a task is measured for each testbed by calling the *perf_counter()* function creating a timestamp. A new call of *perf_counter()* will be performed when the task has been removed from the compute queue which creates a new timestamp for the task. The two timestamps are then subtracted to produce the total completion time for a task. The formula 2 presents the calculations for the total completion time t_T, where t_{list} is the time for requesting and receiving the list of connected nodes, $N(t_{help})$ the time to ask for help from N nodes, t_{data} is the time to send the message containing one half of the task, t_{sort} is the time it takes to sort the task, t_{result} is the time for the result to be sent back to the source node, $t_{finished}$ is the timestamp when a task is completed, $t_{generated}$ is the timestamp when a task is generated.

$$t_T = t_{list} + n(t_{help}) + t_{data} + t_{sort} + t_{result} \Leftrightarrow t_{finished} - t_{generated} \quad (2)$$

Outliers based on the completion time are mathematically removed with the Tukey Interquartile Range method (IQR) presented in [35]. The method determines what data is to be considered an outlier by checking if the value being evaluated by the method is below the Lower Fence (LF) or above the Upper Fence (UF). The IQR is determined by subtracting the upper quartile Q_3 by the lower quartile Q_1, this is presented in formula 3.

$$IQR = Q_3 - Q_1 \quad (3)$$

The LF and UF are calculated using the formulas 4 and 5 accordingly.

$$LF = Q_1 - 1.5 * IQR \quad (4)$$

$$UF = Q_3 + 1.5 * IQR \quad (5)$$

The average completion time and data size is calculated using formula 6, where n is the total number of data points and x_i represents each data point.

$$Average = \bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

(6)

The standard deviation (STDEV) for the completion time and data size is calculated using the formula 7, where s is the STDEV, n is the total number of data points, x_i represents each data point, and $\bar{x}$ is the average completion time or data size.

$$STDEV = s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

(7)

The maximum and minimum are extracted by selecting the highest and lowest value for the completion time and data size respectively.

The scalability is evaluated and measured by using formula 8, where P_{list} is the number of permutations of the list, and N is the total number of nodes in the network.

$$P_{list} = N!, N \geq 0$$

(8)

6 Results

The following sub-chapters describes and presents the results and findings for the different testbeds. The measurements performed for each configuration are completion time for tasks, data size, and scalability.

6.1 Resulting testbeds

Four testbeds have been constructed and implemented for this thesis. The specifications of the devices used can be seen in chapter 4.4 in tables 4 and 5.

The first one being a reference testbed where one RPi is connected to a server (PC) providing greater computational properties than that of an RPi. This testbed can be seen in figure 25.

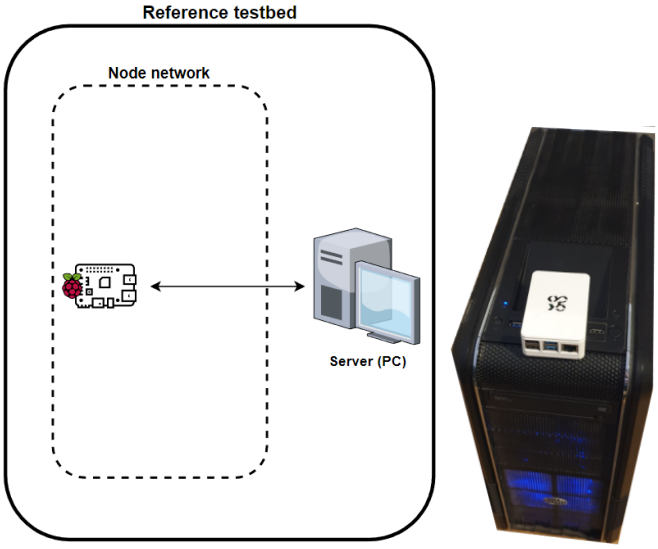

Figure 25: Reference testbed

The second testbed consist of one RPi connected to a coordinator (Macbook Pro). This testbed can be seen in figure 26.

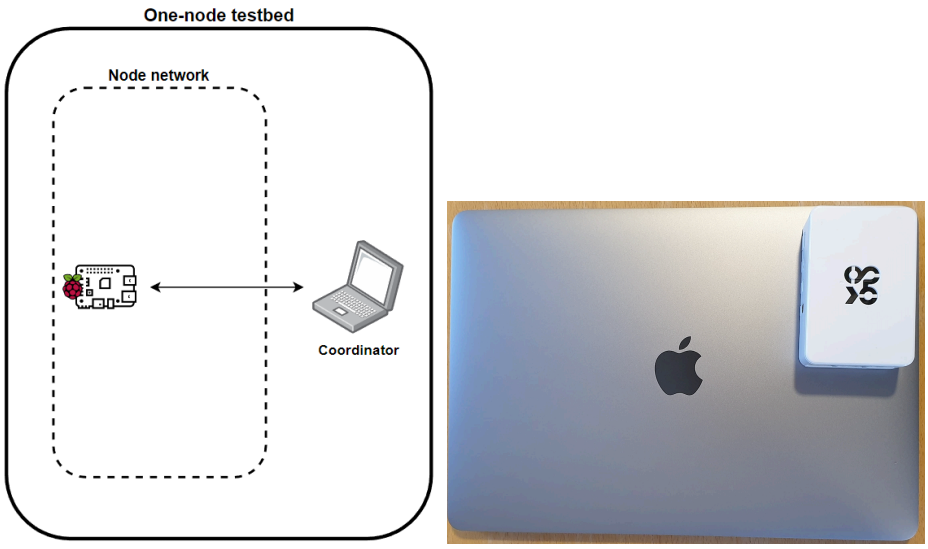

Figure 26: One-node testbed

The third testbed consist of two RPi's connected to the coordinator (Macbook Pro). This testbed can be seen in figure 27.

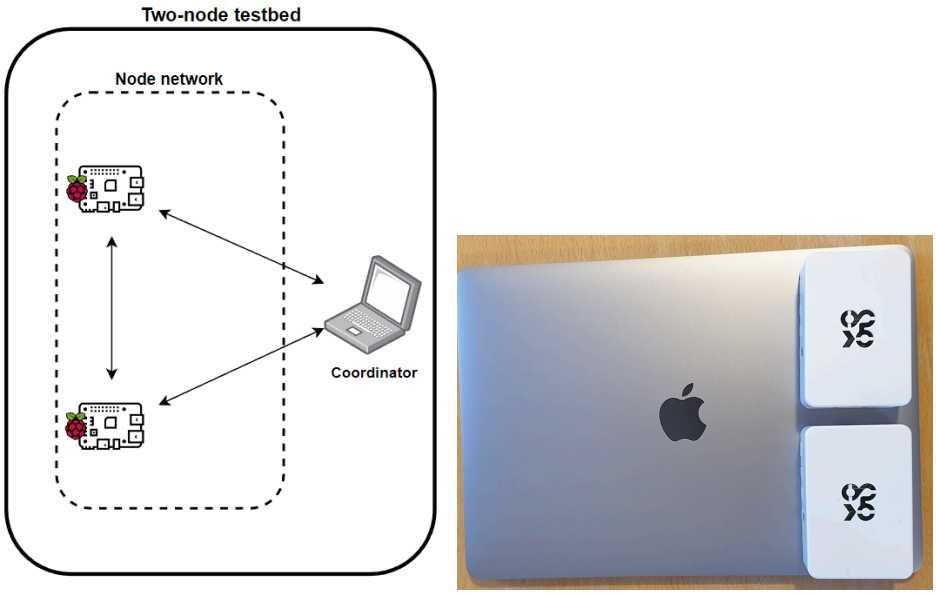

Figure 27: Two-node testbed

The fourth testbed consist of three RPi's connected to the coordinator (Macbook Pro). This testbed can be seen in figure 28.

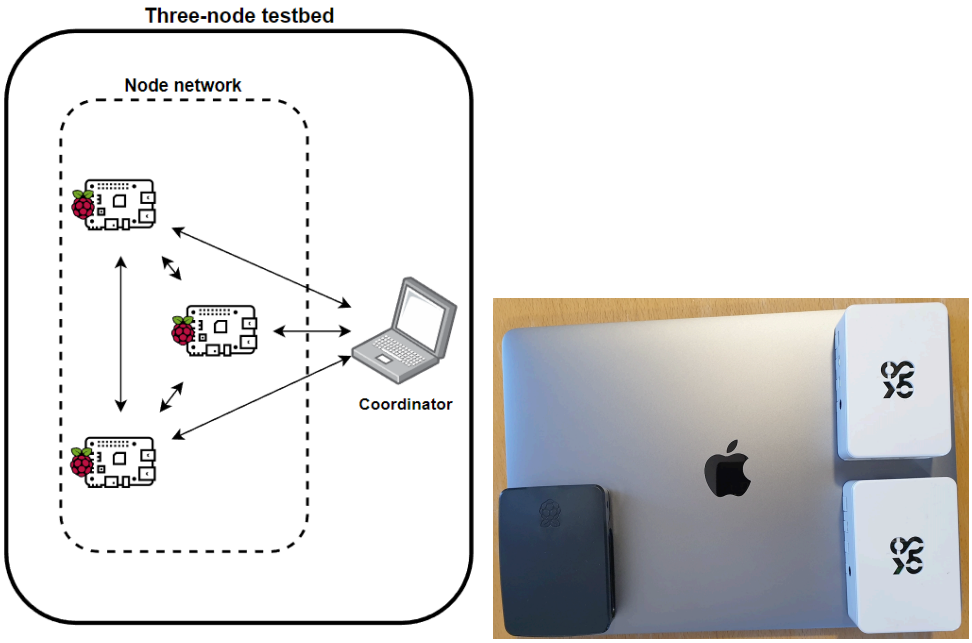

Figure 28: Three-node testbed

6.2 Measurement results – Completion time

This subchapter presents the results from the different testbeds and their configurations regarding completion time for tasks in correspondence with the data size. The threshold referred to for the different testbeds of this subchapter describe the threshold for helping another node with a task and not the threshold for needing help with a task.

6.2.1 Reference testbed

The completion time for all the tasks of the reference testbed including outliers is presented in figure 29 and excluding outliers is presented in figure 30. What can be seen in figure 29 is that there are significant outliers, most visually notable those above the completion time 8000ms. When removing outliers from the data, the algorithm determined that every data point above ca. 5500ms is an outlier providing the result in figure 30. There is no data referring to data not being sent; this is because of the testbed's configuration always sending data to the reference node. For this testbed we can observe a pattern, where we have two linear formations with a close origin ranging from around 100ms to 400ms and also a cluster forming in the middle of these two linear formations ranging from around 1200ms to 2800ms.

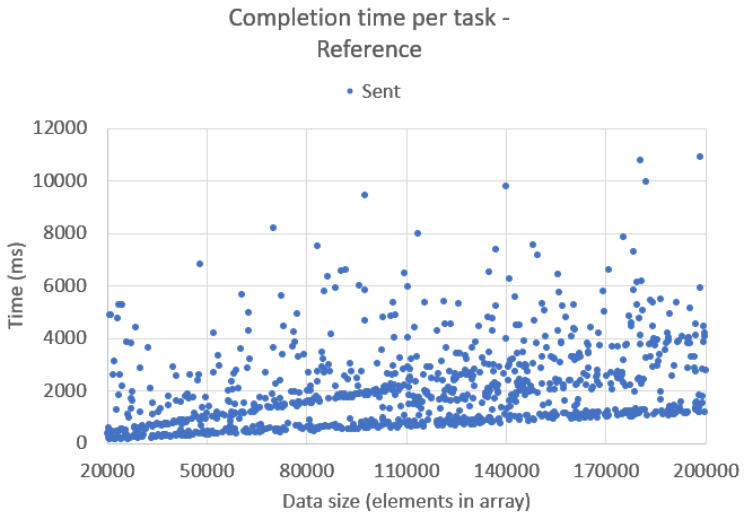

Figure 29: Reference testbed including outliers

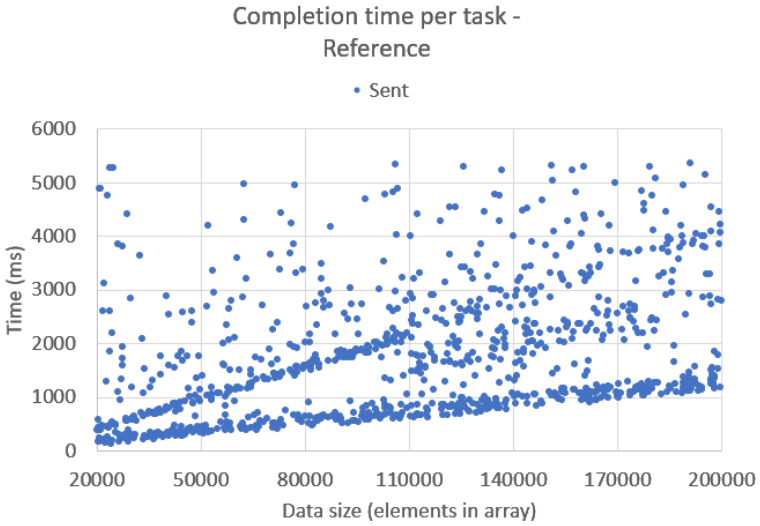

Figure 30: Reference testbed excluding outliers

Table 7 and 8, presents a summary of the task completion time for the reference testbed including and excluding outliers respectively. There are notable differences for the two tables. The exclusion of outliers provided the following results: the average time has lowered by ca. 11%; the STDEV is lowered by ca. 23%; the maximum value is lowered by ca. 51%; the minimum value has not changed.

Table 7: Reference testbed summary including outliers

Summary of completion time for tasks on the reference testbed including outliers (ms)				
Testbed	**Avg**	**STDEV**	**Max**	**Min**
Reference	1960	1580	10920	140

Table 8: Reference testbed summary excluding outliers

Summary of completion time for tasks on the reference testbed excluding outliers (ms)				
Testbed	**Avg**	**STDEV**	**Max**	**Min**
Reference	1750	1210	5370	140

6.2.2 One node

The completion time per task including and excluding outliers for when one node is present in the network is presented respectively in figures 31 and 32. For these figures the outliers are not visually too notable. The maximum value does not exceed 9000ms when including outliers and does not exceed 7500ms. In contrast to the reference testbed, this one-node testbed does not send anything to another node since it is alone in the network, forcing it to compute every task itself. For this testbed we can observe the linear relationship between the completion time and the data size, with some additional scattered data points.

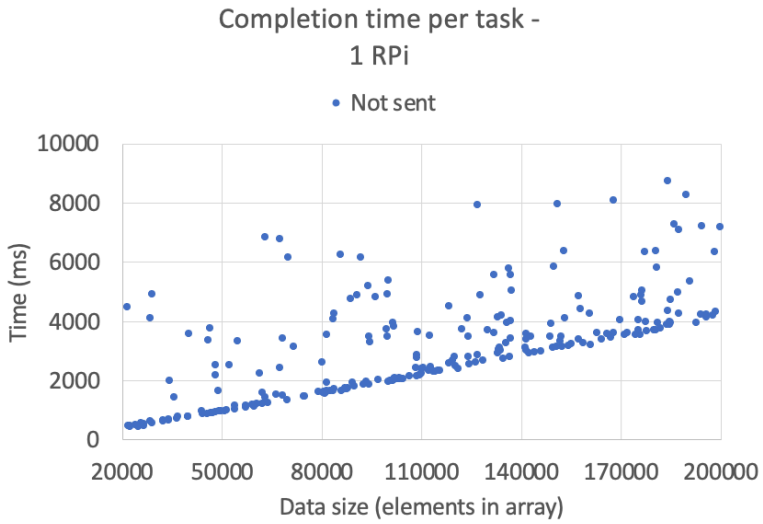

Figure 31: One-node testbed including outliers

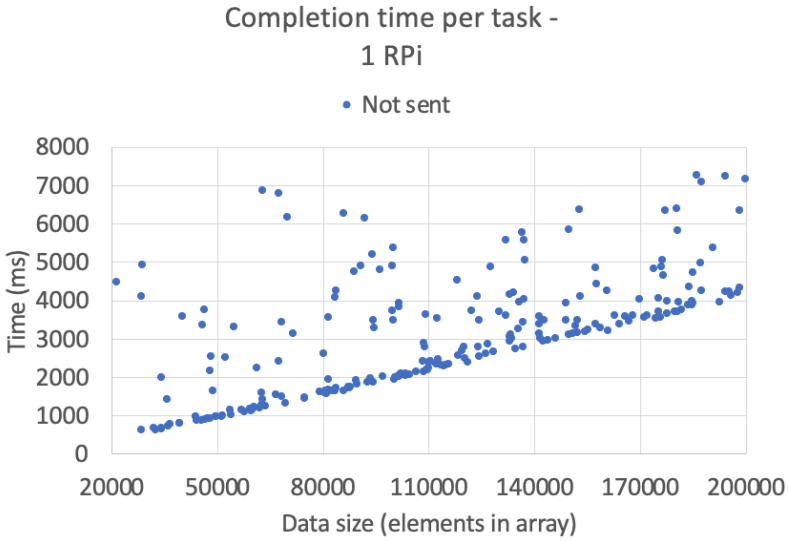

Figure 32: One-node testbed excluding outliers

Figure 33 shows the relationship between the inclusion and exclusion of outliers for this testbed with zero as the threshold and the reference testbed. Multiple thresholds are not present for this testbed since it can't receive help with tasks from other fog nodes. This testbed does not change its average completion time value when excluding like the reference testbed, but only the STDEV has changed. This testbed also performs worse than the reference testbed.

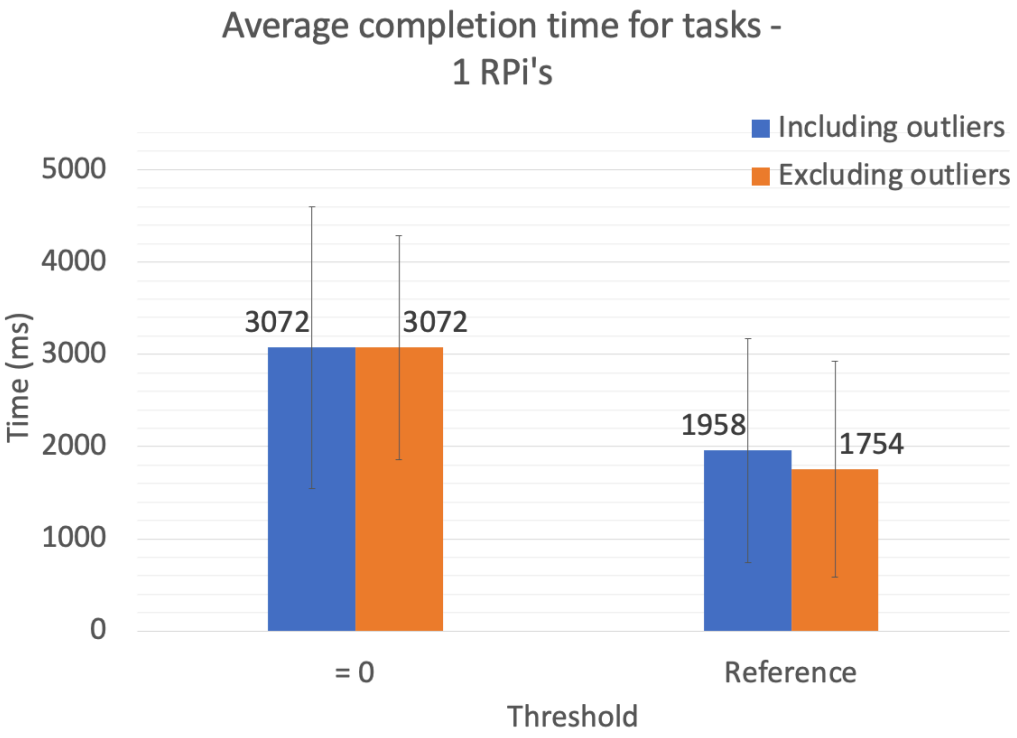

Figure 33: Comparison of the one-node and reference testbeds

Table 9 and 10 presents a summary of completion time for tasks including and excluding outliers for the one-node testbed in comparison to the reference testbed. For the difference of the table presenting the inclusion of outliers there is the following differences with focus on the one-node testbed: average completion time is around 36% higher; the STDEV is around 9% higher; the maximum completion time is around 20% lower; the minimum value is around 68% higher. When excluding outliers, the differences change to the following: the average completion time is around 43% higher; the STDEV is around 20% higher; the maximum value is around 26% higher; the minimum value is around 78% higher.

Table 9: Summary of the one-node and reference testbeds including outliers

Summary of completion time for tasks for 1 RPi including outliers (ms)				
Testbed	**Avg**	**STDEV**	**Max**	**Min**
One-node	3070	1730	8760	440
Reference	1960	1580	10920	140

Table 10: Summary of the one-node and reference testbeds excluding outliers

Summary of completion time for tasks for 1 RPi excluding outliers (ms)				
Testbed	**Avg**	**STDEV**	**Max**	**Min**
One-node	3070	1520	7280	640
Reference	1750	1210	5370	140

6.2.3 Two nodes

The completion time per task including and excluding outliers for when two nodes are present in the network with threshold zero is presented respectively in figures 34 and 35. What can be seen here is that the outliers seem to range from around 6500ms and up to 11000ms when comparing the two figures. What is most notable is the difference in the completion time for when a task has been sent at the data size around 110000 and up to 200000. While the data not being sent is forming a linear relationship, the data being split and sent is dropping in completion time where also the differences of completion time increase with the data size. In both two figures the majority of tasks is still processed on the source

node due to the threshold for helping another node being the lowest value of the testbed configurations.

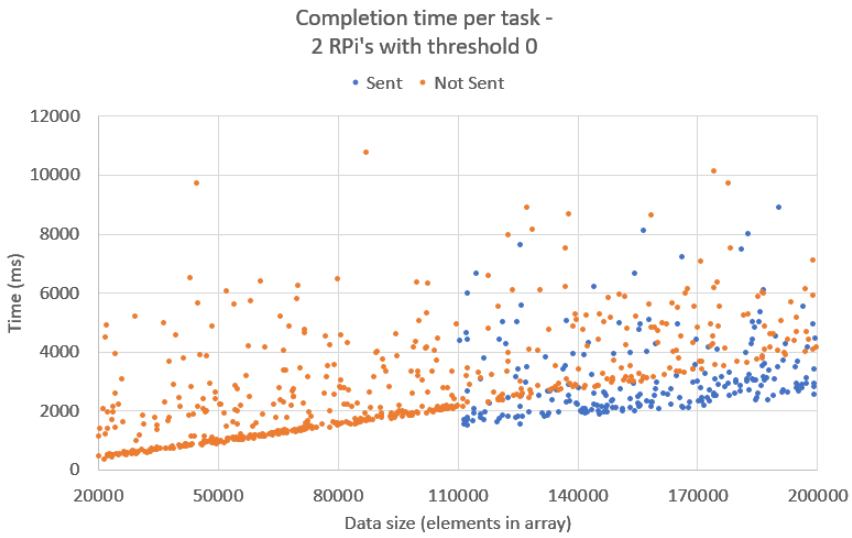

Figure 34: Two-node testbed with threshold zero including outliers

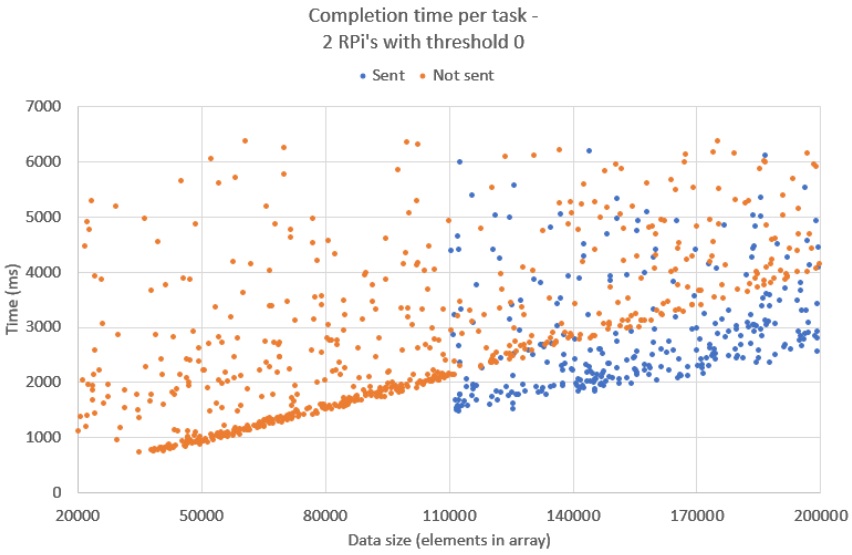

Figure 35: Two-node testbed with threshold zero excluding outliers

The completion time per task including and excluding outliers for when two nodes are present in the network with threshold one is presented respectively in figures 36 and 37. What can be seen here is that the outliers seem to have the around the same range as the two previous figures but with a small increase. As also seen in figures for the previous threshold configuration is the decrease of completion time exceeding the

data size 110000. There also seems to be a linear relationship for this threshold as well. In contrast to the previous threshold, this threshold increases the number of tasks being split and sent, with a few tasks not being sent when exceeding the data size 110000.

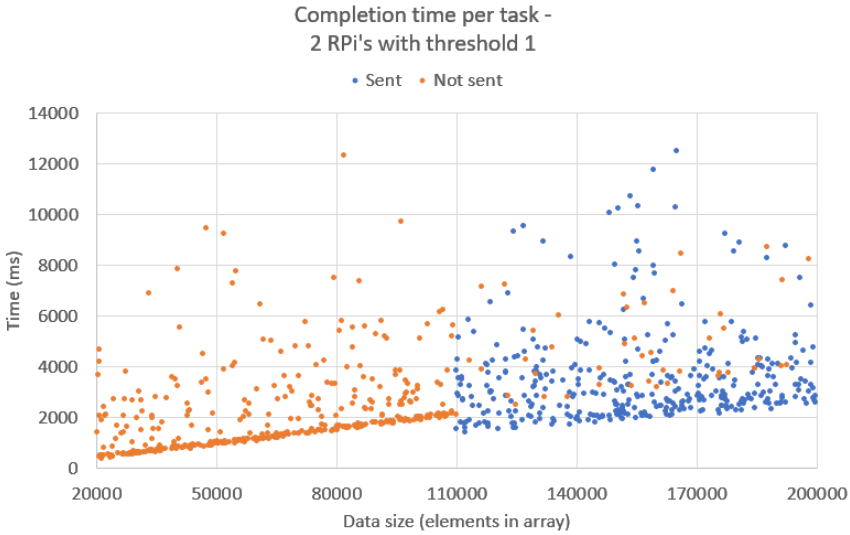

Figure 36: Two-node testbed with threshold one including outliers

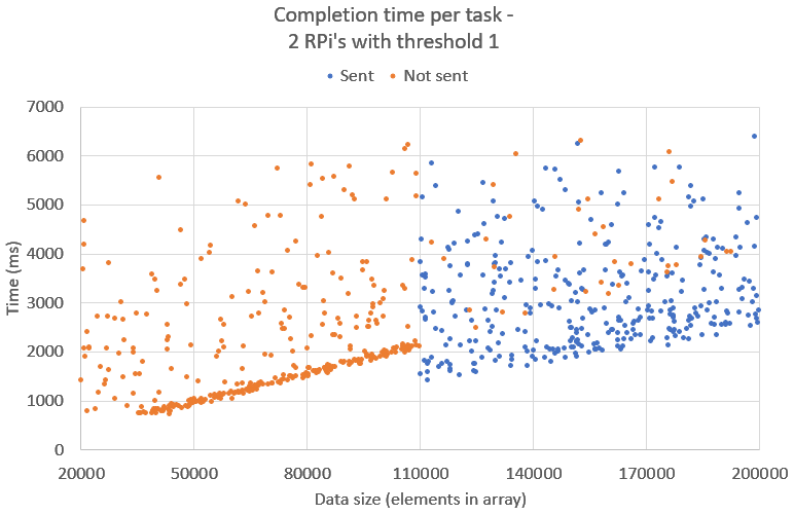

Figure 37: Two-node testbed with threshold one excluding outliers

The completion time per task including and excluding outliers for when two nodes are present in the network with threshold two is presented respectively in figures 38 and 39. This threshold seems to provide the same pattern as threshold one with the main difference being the number

of not sent tasks being even lower when exceeding 110000 for this threshold.

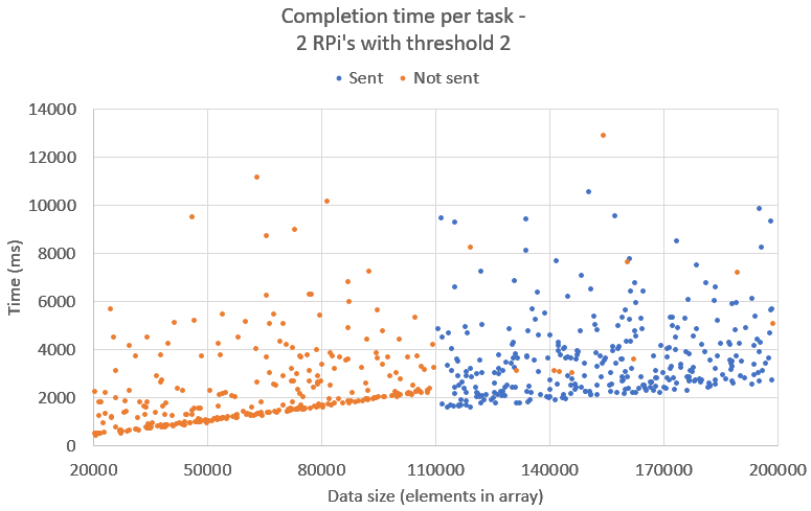

Figure 38: Two-node testbed with threshold two including outliers

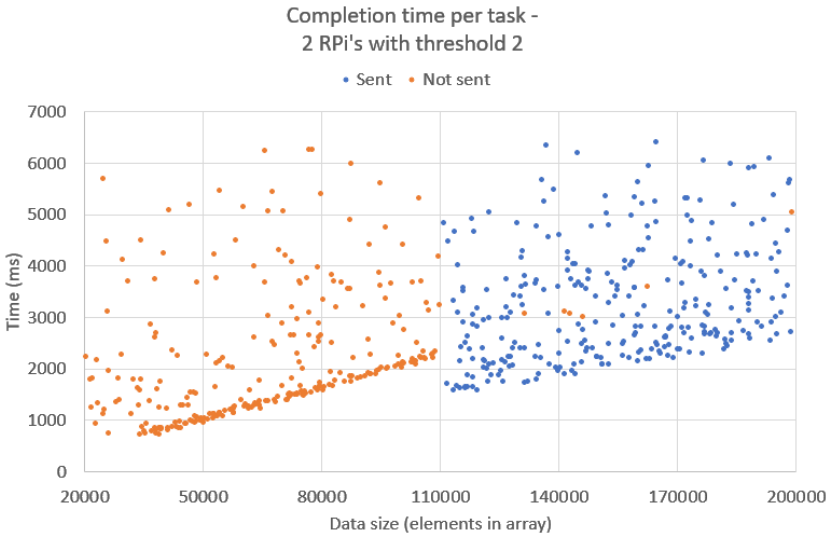

Figure 39: Two-node testbed with threshold two excluding outliers

The completion time per task including and excluding outliers for when two nodes are present in the network with threshold four is presented respectively in figures 40 and 41. As the with the previous threshold, this threshold provides the same pattern again but with all tasks exceeding a data size around 110000 being sent.

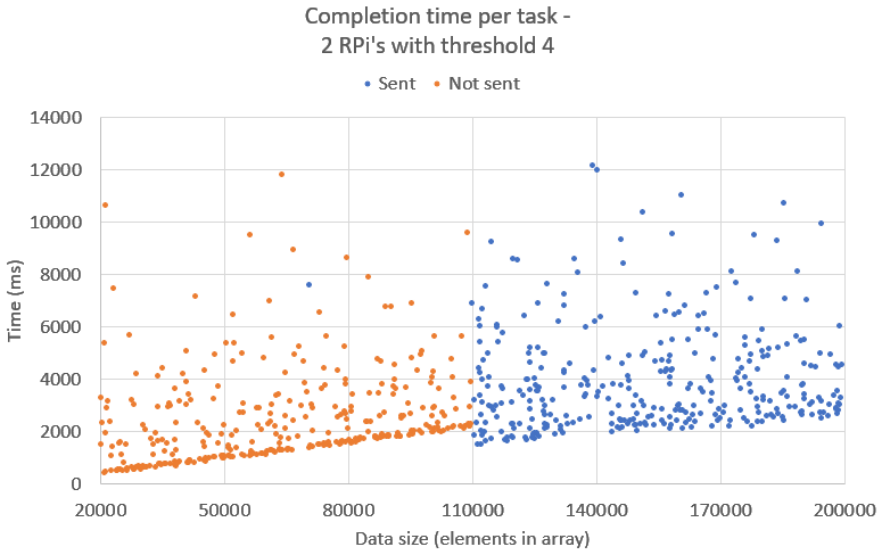

Figure 40: Two-node testbed with threshold four including outliers

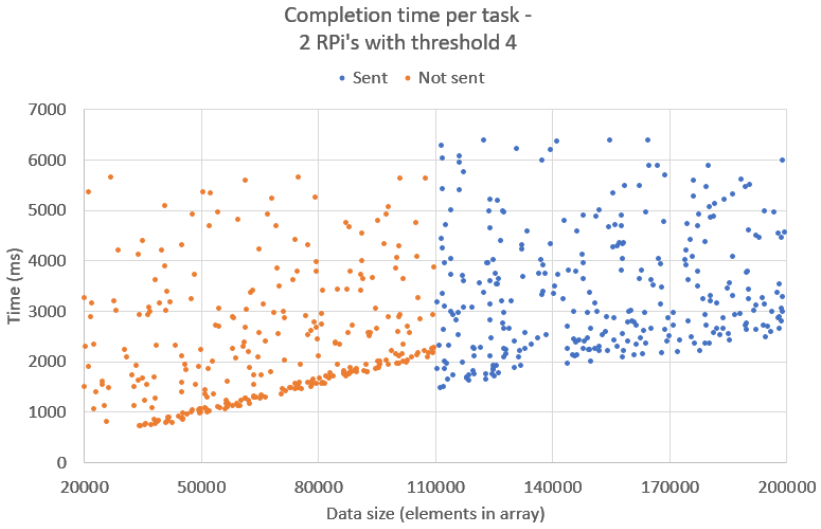

Figure 41: Two-node testbed with threshold four excluding outliers

Figure 42 shows the relationship between the inclusion and exclusion of outliers for the two-node testbed with all the corresponding thresholds and the reference testbed. The threshold zero does not provide a difference in the average completion time for the inclusion and exclusion of outliers. However, for the remaining thresholds the difference of the average completion time has decreased when excluding the outliers. What is also visualized in this figure is that this testbed performs worse than the reference testbed where threshold zero performs the best out of these with a small difference.

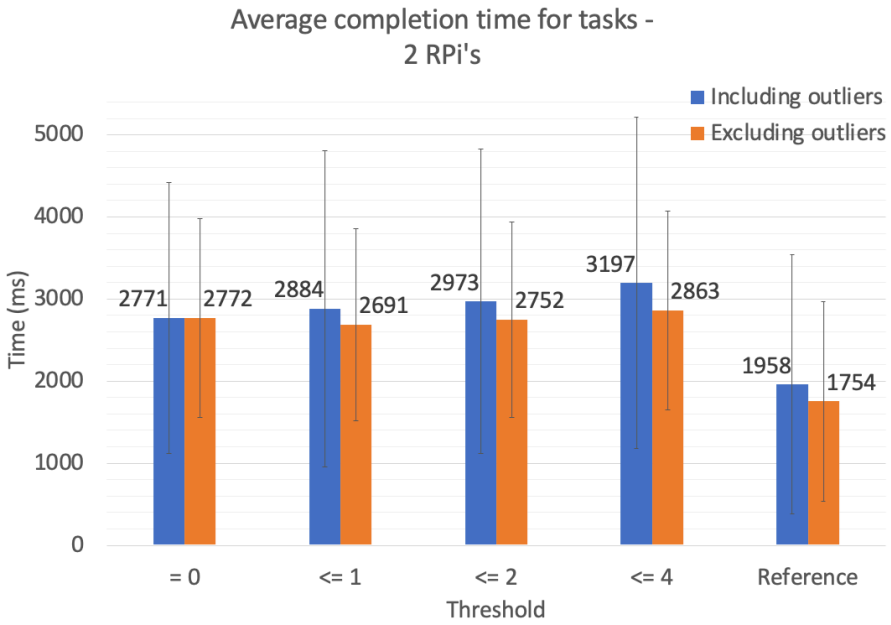

Figure 42: Comparison of the two-node and reference testbeds

Table 11 and 12 presents a summary of completion time for tasks including and excluding outliers for the two-node testbed in comparison to the reference testbed. For the difference of the table presenting the inclusion of outliers there is the following differences with focus on the best performing threshold of the two-node testbed: average completion time is around 29% higher; the STDEV is around 4% higher; the maximum completion time are the same; the minimum value is around 60% higher. When excluding outliers, the differences change to the following: the average completion time is around 37% higher; the STDEV is the same; the maximum value is around 14% higher; the minimum value is around 87% higher.

Table 11: Summary of the two-node and reference testbeds including outliers

Summary of completion time for tasks for 2 RPi's including outliers (*ms*)				
Threshold	**Avg**	**STDEV**	**Max**	**Min**
= 0	2770	1650	10920	340

<= 1	2880	1930	12510	350
<= 2	2970	1850	12890	410
<= 4	3200	2010	12140	410
Reference	1960	1580	10920	140

Table 12: Summary of the two-node and reference testbeds excluding outliers

Summary of completion time for tasks for 2 RPi's excluding outliers (*ms*)				
Threshold	**Avg**	**STDEV**	**Max**	**Min**
= 0	2770	1210	6270	1100
<= 1	2690	1170	6340	1070
<= 2	2750	1190	4130	1150
<= 4	2860	1210	6020	1100
Reference	1750	1210	5370	140

6.2.4 Three nodes

The completion time per task including and excluding outliers for when three nodes are present in the network with threshold zero is presented respectively in figures 43 and 44. This threshold seems to visually be in par with the same threshold for the two-node testbed when observing the number of split and sent tasks, and the drop in completion time for the split and sent tasks.

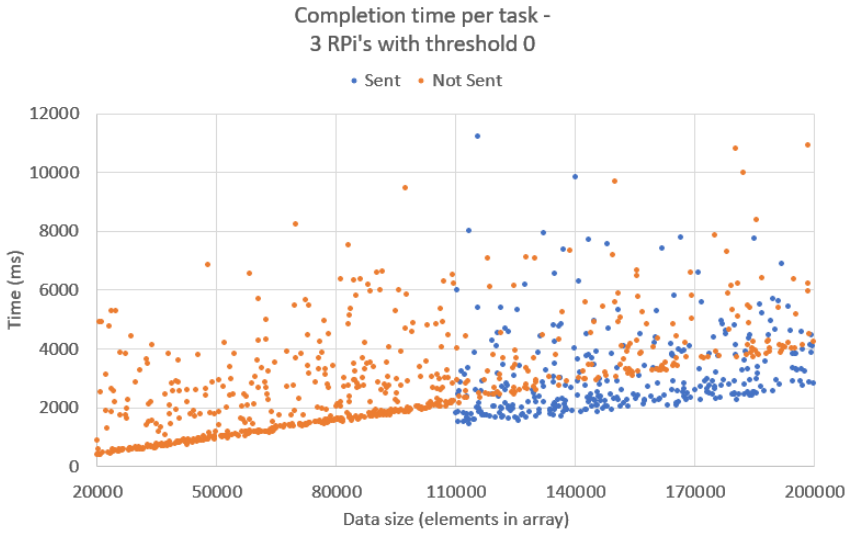

Figure 43: Three-node testbed with threshold zero including outliers

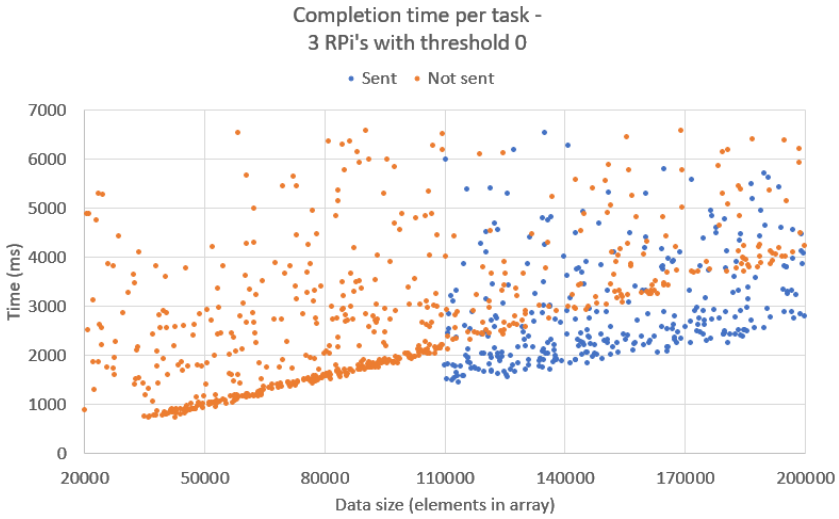

Figure 44: Three-node testbed with threshold one excluding outliers

The completion time per task including and excluding outliers for when three nodes are present in the network with threshold one is presented respectively in figures 45 and 46. For this threshold there is a notable difference between the inclusion and exclusion of outliers. What can be seen for the inclusion of outliers is that the completion time extends over 25000ms in the most extreme case which is the highest completion time yet. The exclusion of the outliers pulls down the maximum completion time to under 7000ms. As with the previous testbeds the number of split and sent tasks are the majority of the data points when exceeding 110000 in size with a few tasks not being split and sent.

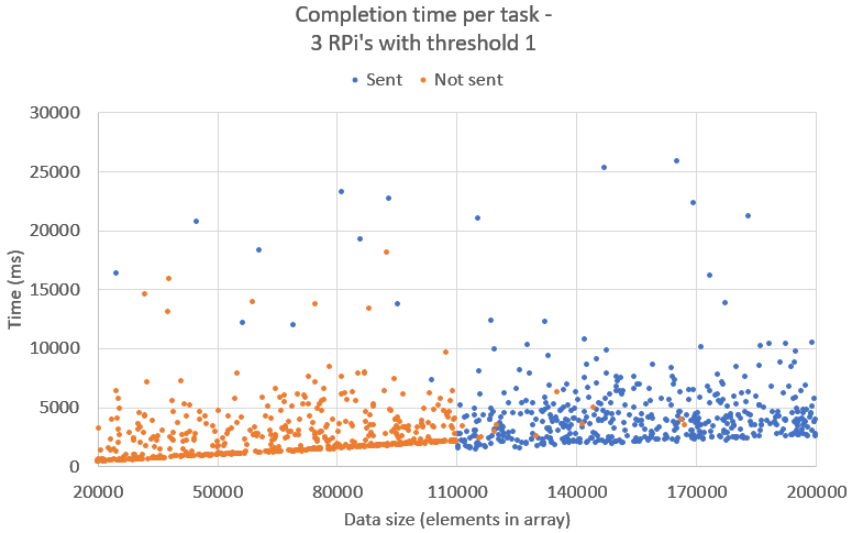

Figure 45: Three-node testbed with threshold one including outliers

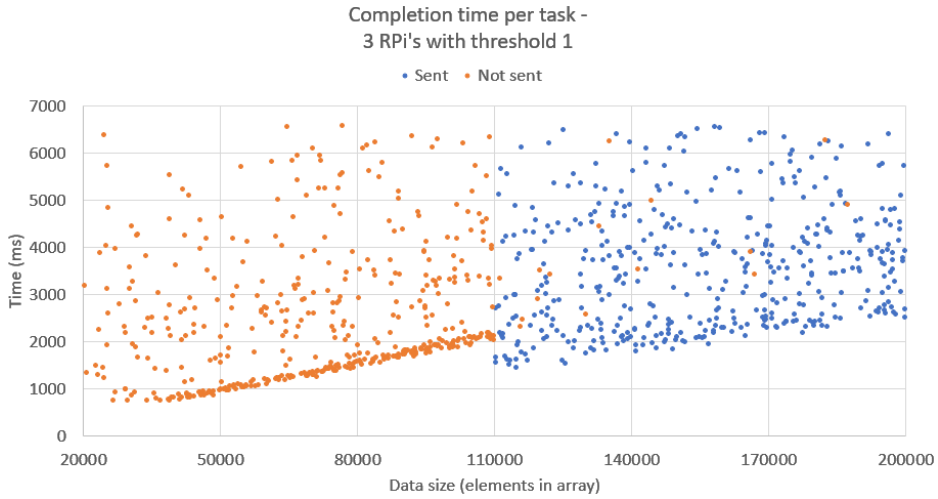

Figure 46: Three-node testbed with threshold one excluding outliers

The completion time per task including and excluding outliers for when three nodes are present in the network with threshold two is presented respectively in figures 47 and 48. Once again there appears extreme outlier data points but they are not as large in completion time as the previous threshold. This threshold follows the pattern of data being split and sent which previous thresholds have provided as well.

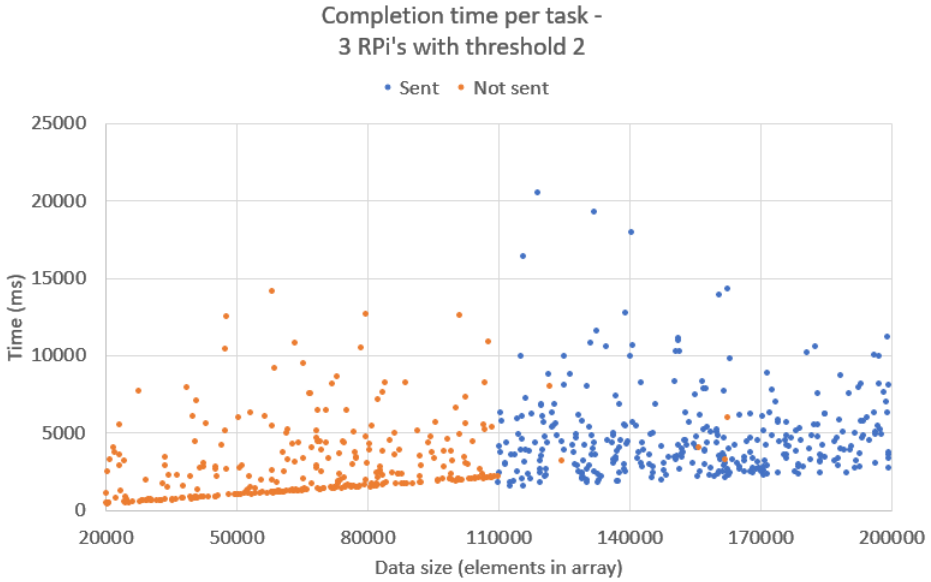

Figure 47: Three-node testbed with threshold two including outliers

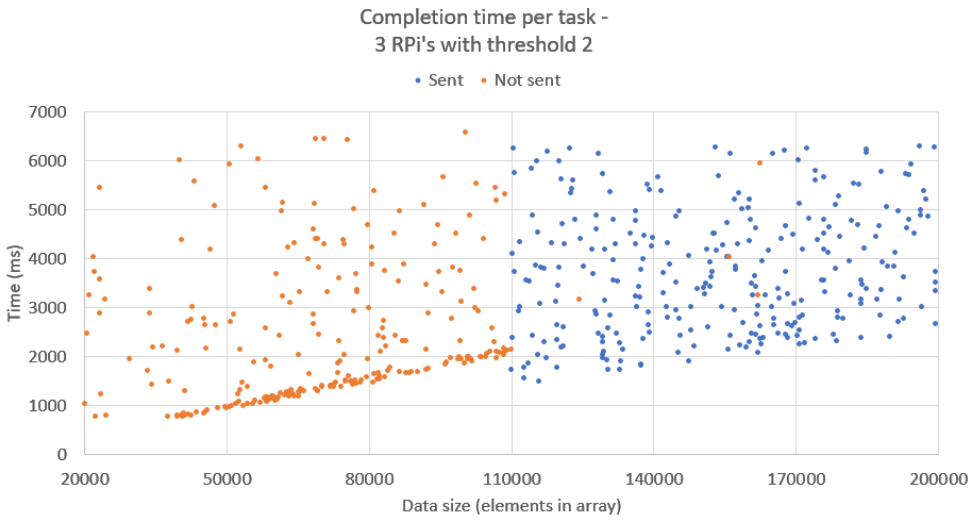

Figure 48: Three-node testbed with threshold two excluding outliers

The completion time per task including and excluding outliers for when three nodes are present in the network with threshold four is presented respectively in figures 49 and 50. As with threshold one and two the outliers are quite extreme but does in this case not exceed 20000ms. The same pattern appears here with the number of split and sent tasks with the increase of threshold for the testbeds.

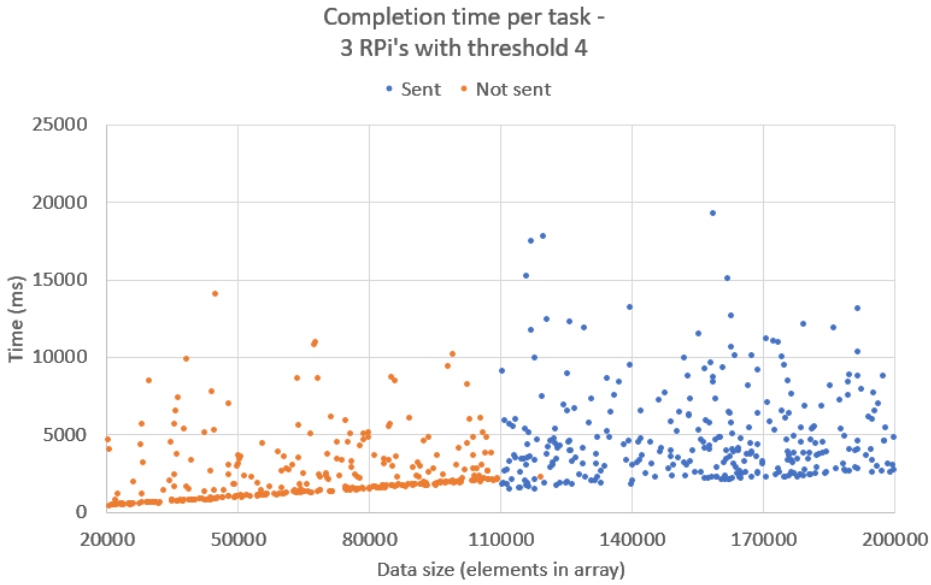

Figure 47: Three-node testbed with threshold four including outliers

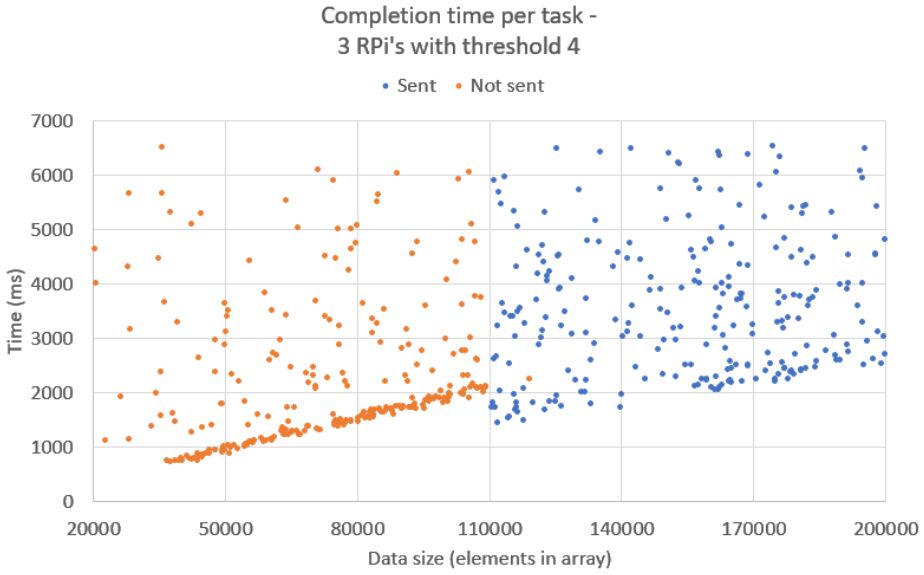

Figure 48: Three-node testbed with threshold four excluding outliers

Figure 51 shows the relationship between the inclusion and exclusion of outliers for the three-node testbed with all the corresponding thresholds and the reference testbed. For this testbed the best performing threshold is zero with a bigger difference to the other thresholds. This testbed and its configurations also perform worse than the reference testbed.

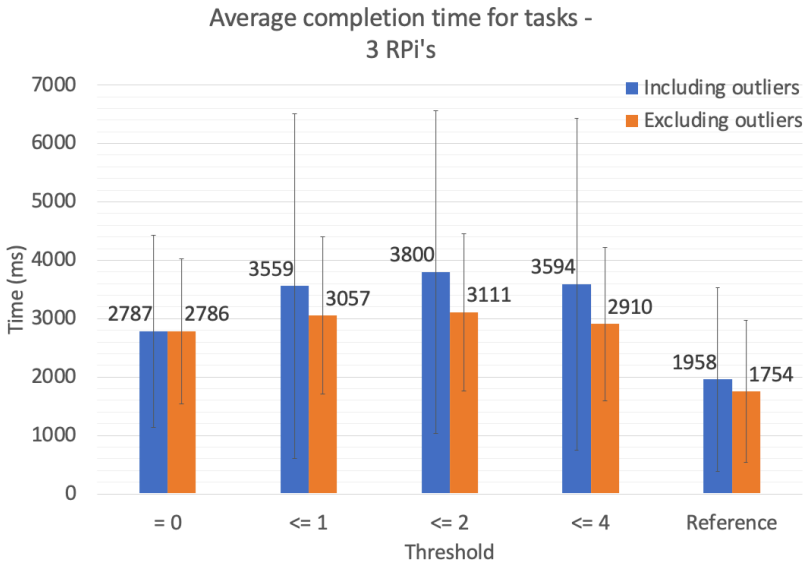

Figure 49: Comparison of the three-node and reference testbeds

Table 13 and 14 presents a summary of completion time for tasks including and excluding outliers for the one-node testbed in comparison to the reference testbed. For the difference of the table presenting the inclusion of outliers there is the following differences with focus on the three-node testbed: average completion time is around 29% higher; the STDEV is around 4% higher; the maximum completion time is around 3% lower; the minimum value is around 63% higher. When excluding outliers, the differences change to the following: the average completion time is around 37% higher; the STDEV is around 2% higher; the maximum value is around 18% higher; the minimum value is around 87% higher.

Table 13: Summary of the three-node and reference testbeds including outliers

Summary of completion time for tasks for 3 RPi's including outliers (*ms*)				
Threshold	**Avg**	**STDEV**	**Max**	**Min**
= 0	2780	1650	11200	380
<= 1	3560	2950	25890	350

<= 2	3800	2770	20520	380
<= 4	3590	2840	19230	380
Reference	1960	1580	10920	140

Table 14: Summary of the three-node and reference testbeds excluding outliers

Summary of completion time for tasks for 3 RPi's excluding outliers (*ms*)				
Threshold	**Avg**	**STDEV**	**Max**	**Min**
= 0	2780	1240	6550	1080
<= 1	3060	1350	6560	1080
<= 2	3110	1350	4750	1120
<= 4	2910	1310	6510	1070
Reference	1750	1210	5370	140

6.3 Measurement results – Data size

This sub-chapter present the findings and results from the data sizes of each configuration. The inclusion and exclusion of outliers for the data size is determined by the outlier algorithm being applied on specifically the completion time.

6.3.1 Reference

Table 15 presents a summary of the data size for the reference testbed including and excluding outliers, respectively. The differences here are almost non-existing but for the average data size being 0.9% larger and the STDEV being 0.2% lower when including outliers.

Table 15: Summary reference testbed including outliers

Summary of data size for tasks for the reference testbed including outliers (elements in array)			
Avg	**STDEV**	**Max**	**Min**
110720	51680	199950	20040

Table 16: Summary reference testbed excluding outliers

Summary of data size for tasks for the reference testbed excluding outliers (elements in array)			
Avg	**STDEV**	**Max**	**Min**
109750	51800	199950	20040

6.3.2 One node

Table 17 and 18 presents a summary of the data size for tasks when one node is operating in the network in comparison to the reference testbed. Here it can also be seen that the differences of values are barely any, with the exception of the STDEV when excluding outliers, is around 9% lower for the one-node testbed.

Table 17: Summary of the one-node and reference testbeds including outliers

Summary of data size for tasks including outliers for 1 RPi (elements in array)				
Testbed	**Avg**	**STDEV**	**Max**	**Min**
One-node	109970	49650	199700	21610
Reference	110720	51680	199950	20040

Table 18: Summary of the one-node and reference testbeds excluding outliers

Summary of data size for tasks excluding outliers for 1 RPi (elements in array)				
Testbed	**Avg**	**STDEV**	**Max**	**Min**
One-node	112450	47270	199700	20040
Reference	109750	51800	199950	20040

6.3.3 Two nodes

Table 19 and 20 presents a summary of the data size for tasks when two nodes are operating in the network with their corresponding threshold in comparison to the reference testbed. Here there is a significant difference when including and excluding the outliers of the data set. The average, maximum and minimum values of the data size remain around the same, but the STDEV changes drastically for the exclusion of outliers. The STDEV becomes: 30% lower for threshold zero; 42% lower for threshold one; 50% lower for threshold two; 51% lower for threshold four.

Table 19: Summary of the two-node and reference testbeds including outliers

Summary of data size for tasks including outliers for 2 RPi's (elements in array)				
Threshold	**Avg**	**STDEV**	**Max**	**Min**
= 0	110320	51930	199910	20290
<= 1	109730	52270	199900	20270
<= 2	110250	51930	198960	20510

<= 4	108070	51300	199580	20200
Reference	110720	51680	199950	20040

Table 20: Summary of the two-node and reference testbeds excluding outliers

Summary of data size for tasks excluding outliers for 2 RPi's (elements in array)				
Threshold	**Avg**	**STDEV**	**Max**	**Min**
= 0	126090	36290	199910	20340
<= 1	115830	30080	199900	20270
<= 2	111180	25970	198960	20610
<= 4	109700	25660	199580	20200
Reference	109750	51800	199950	20040

6.3.4 Three nodes

Table 21 and 22 presents a summary of the data size for tasks when three nodes are operating in the network with their corresponding threshold in comparison to the reference testbed. For this testbed the result is similar to the two-node testbed and its corresponding thresholds where the STDEV provides the greatest differences. The STDEV becomes for this testbed: 32% lower for threshold zero; 49% lower for threshold one; 52% lower for threshold two; 53% lower for threshold four.

Table 21: Summary of the three-node and reference testbeds including outliers

Summary of data size for tasks including outliers for 3 RPi's (elements in array)				
Threshold	**Avg**	**STDEV**	**Max**	**Min**
= 0	108590	50180	199950	20040
<= 1	110600	52090	199990	20100
<= 2	112060	50510	199640	20150
<= 4	111000	51220	199960	20590
Reference	110720	51680	199950	20040

Table 22: Summary of the three-node and reference testbeds excluding outliers

Summary of data size for tasks excluding outliers for 3 RPi's (elements in array)				
Threshold	**Avg**	**STDEV**	**Max**	**Min**
= 0	121650	35000	199950	20210
<= 1	114640	26590	199990	20410
<= 2	112350	25110	199520	20160
<= 4	113470	24460	199960	20590
Reference	109750	51800	199950	20040

6.4 Measurement results – Scalability

The scalability is dependent on the threshold for helping other nodes as well as the list of connected nodes that each node receives from the coordinator when needing help. Every time a node gets asked by another node to help with a task, that node will be accepted as a new connection. Depending on the threshold for helping other nodes set for the testbed the connection will either be closed due to the threshold having been reached previously or kept open until the task has been completed by the helping node. The number of simultaneous connections on a single node where help is granted will not exceed the threshold. However, the number of connections on a single node where help is not granted can in contrast be more than that of the threshold. What acts as a counter to this is that each connection will disconnect the moment after a decline has been sent from the potentially helping node, negating the potential scenario where a large number of nodes is connected to a single node.

The order of all connected nodes in the list received from the coordinator is randomly sorted for each new connection made to the coordinator. This provides the network with further scalable properties since a single node will not be overwhelmed by new connections due to the randomization of the list. The testbeds and their configurations evaluated and measured in sub-chapters 6.2 and 6.3 provides the results of the permutations presented in table 22 when using formula 8 provided in chapter 5.5. The reference testbed and all the corresponding thresholds are not included since they do not impact the number of permutations of the list.

Table 23: Permutations of the list of connected nodes

1 RPi	**2 RPi's**	**3 RPi's**	**n RPi's**
$P_{list} = 1$	$P_{list} = 2$	$P_{list} = 6$	$P_{list} = n!$

Due to the properties of the threshold for helping other nodes, the fast disconnects and number of permutations increasing with the increase of nodes in the network, a single node will not be overwhelmed by different connections.

7 Discussion

This chapter presents an analysis and discussion of the results, the project method, the scientific aspects, and lastly ethical and societal considerations.

7.1 Analysis and discussion of results

As presented in sub-chapter 6.3 the results for the completion of tasks for the two-node and three-node testbeds with threshold zero for when helping other nodes are almost the same. The difference is 10ms in favor of the two-node testbed when including and excluding outliers, i.e., a 0.4% difference. On the other hand, the three-node testbed is performing 290ms better than the one-node testbed, i.e., a 9.5% difference. These differences are presented in table 24.

Table 24: Testbed differences in completion time

Outliers	**1 RPi (ms)**	**2 RPi's (ms)**	**3 RPi's (ms)**
Including	3070	2770	2780
Excluding	3070	2770	2780

The similarity of results regarding the threshold zero might be because of the choices made on the threshold for needing help and the threshold for helping others. As mentioned in chapter 4.4 the threshold for needing help is not changed for the different testbeds but the threshold for helping others is. The performance of the different testbeds could possibly have been different if the threshold for needing help is changed instead or choosing to observe CPU or RAM usage or both. If the threshold for needing help is changed the drop would appear before or after the data size 110000 which could have resulted in either a positive effect on the average completion time for tasks or a negative effect. However, if the choice of CPU or RAM usage had been made an issue still arises. The issue is that the reading of the CPU and RAM usage will not be performed constantly, which does not lead to accurate decisions to be made for when nodes need or giving help. This issue is especially true since the testbeds are not being run on true multithreading as

mentioned in chapter 5. Further expanding on the non-true multithreading property of the implementations, the outliers that are present in the results could be an effect of this property. This could have been countered by using another Python implementation than CPython, as mentioned in chapter 5.1 or by using the *multiprocessing* library [36] which side-steppes the GIL which uses subprocesses instead of threads. The use of this library has a requirement however, which is that it can only take advantage of the multiprocessing properties on a multiprocessor device. Even though the GIL is side-stepped, it has functions to create locks for shared resources. When locks are not used there needs to be great care when working with shared resources since readers-writers' problems can occur if caution is not taken.

The most notable results for the completion time of tasks are the drastic drops of completion time for tasks that are split and shared. This property is a good indicator that the splitting and distribution of a task is desirable. When comparing two data points, one not sent and one sent, with the data sizes 78354 and 110742 respectively, they perform basically the same in regard to completion time, 1493ms and 1496ms respectively, even though the tasks have a 30% difference in data size. The pattern of this performance can be recognized for both the two-node and three-node testbeds on all the different threshold configurations. It would be interesting to see how the completion time differs in a larger testbed and how changing the threshold for when help is needed would affect it instead of changing the threshold for when helping another node.

Since the communication is done through TCP, a general partly non-controllable factor that impacts the results is the WiFi which the testbeds were set up on. This might provide an inconsistency for the completion time for the tasks. In regard to the communication and transfer of data and its size through the connections of the network, the results do not indicate that the differences of data sizes impact the transfer time a large amount. If the size of the data would have been larger, the impact of it could have made a bigger difference but in the testbeds implemented for this thesis there is no indication of it.

The scalability of the testbeds seems to not be an issue for the two and three-node testbeds with the threshold for helping others is zero. When increasing the threshold for helping others the three-node testbed does seem to have an impact on the average completion time for tasks which

could be scalability related. However, to verify this, more nodes need to be implemented before a more definitive result could be provided. The theoretical aspects do indicate that the scalability regarding the randomized list of connected nodes is impacted positively.

7.2 Project method discussion

The method was in retrospect an appropriate way to conduct the work of this thesis. The phases of the method made it possible to structure the work to be performed into milestones which made it easy to determine if the research questions are being answered.

Phase one of the project method with its four milestones have been completed. More than five related works have been studied, which are incorporated mainly to chapters 1-4. Five challenges of the traditional cloud computing architecture have been provided namely, security, reliability, storage, latency, and monopoly, covered in chapter 1 and 2. Four research questions and the scientific contribution of this thesis have been stated in chapter 1. This phase had no direct challenges due to there being a lot of research having been done in the area of mist, fog, and cloud computing by other people.

Phase two of the project method and its three milestones have been completed. The definition of the scope of the thesis has been provided in chapter 1 where it is stated the main focus will lie on designing and implementing a novel architecture model for future IoT applications. The definitions of each milestone can be seen in chapter 3. The task defined for the testbed was explained and defined in chapter 4 to be a sorting task, using the sorting algorithm merge sort. The evaluation areas were defined to be completion time for tasks, data size, and scalability. The challenge of this phase was to choose and define the different evaluation areas, completion time, data size, and scalability. The choice of evaluation areas could have been made different by choosing other properties such as measuring the transfer time of data, actual computation time for the tasks and not the time it takes for a task to be completed. This would then require the implementation of the testbeds to be changed accordingly. A reiteration of the phase could have possibly improved it but only after the implementation had been done.

Phase three and its four milestones were achieved and described in detail in chapter 5. The mist layer became a simulation of a sensor generating

data in the form of arrays on the RPi/RPi's incorporated into the fog layer operating on the devices. The cloud layer was implemented as a standalone reference testbed where one RPi was connected to a server computer in the form of a PC which significantly larger processing capabilities than that of the fog nodes. The challenges of this phase were designing and implementing the fog layer. An issue here was the lack of technical knowledge for this type of network. A decision that could have been made differently is the programming language, namely using Java instead of Python, where the multithreading properties of Java could have been taken advantage of. This does however come with the drawback of requiring more installations to be made on the RPi's for it to be able to be run. This phase could have been improved by reiterating it since the technical knowledge has grown since the phase ended.

Phase four did not consist of multiple different milestones but had only the goal of demonstrating the developed testbeds for the supervisor. This phase was completed as well.

Like the previous phase, phase five did not have multiple different milestones either and was fulfilled by the quantitively measurements of completion time for tasks, data size, and scalability where the results of these are provided in chapter 6. The challenge for this phase was the large amount of different data that was to be extracted and analyzed. This is because there are four different testbeds, where two of them have three different configurations regarding the threshold.

Phase six and its goal is fulfilled by this report and the presentation that follows.

7.3 Scientific discussion

The DSRM proved to be great method for this thesis. It provided relevant and supporting steps in order to thoroughly investigate, implement, and evaluate the overall work. The possibility of further breaking down the different steps or phases into smaller milestones extends the viability of the DSRM in this type of research.

As generally gained knowledge from this thesis is that fog computing is a concept on the rise and could provide useful in future IoT applications. More specified gained knowledge is the potential fog computing has for when tasks can be split and distributed for computation or processing on

other nodes. The splitting of tasks was not present in the three related works presented in chapter 2. In those works, the whole tasks were distributed to other nodes. Another thing all the related works have in common that differentiates them from the implementation performed in this thesis is simulating their proposed models or algorithms. The related works do provide algorithms for offloading which also plays a large role in the collaboration of nodes and could be tested or used in this thesis. This is especially true for *Hierarchical fog-cloud computing for IoT systems: a computation offloading game* since their work clearly performs better with the increase of fog nodes. The requirement for adopting their work into this thesis would be the amount of fog nodes present in the testbeds.

7.4 Ethical and societal discussion

The data managed in this thesis is randomly generated and is not personal or in need of being encrypted by any means. However, in the larger perspective where personal or sensitive data is processed and overseen by fog nodes the care for security and privacy needs to be a priority. This is especially true for big data applications such as those presented by Al Nuaimi *et al*. [37].

The price of fog services could potentially be lower than that of cloud services, much because the companies that offer cloud services have a monopoly on this technology due to the cost of running and administrating these. This could be the case if future fog services can be proven to become an open-source technology that smaller companies could adopt and manage.

This thesis emphasizes the use of of-the-shelf-hardware as its processing power which could lead to a potential gateway into reusing hardware alleviating negative environmental effects. Even though the technological advancements are progressing in a fast pace the use for "old" hardware acting as fog nodes as explored by Corotinschi and Găitan [38].

8 Conclusions

The first research question *Why and to what extent is distributed pooling of resources needed for future IoT applications?* was fulfilled through milestone M1.4 from phase one which was the literature study. The literature study covered multiple challenges with the traditional cloud computing model. These are presented in chapter 1 and 2 as security, reliability, storage, latency, and monopoly.

The second research question *How can multiple IoT-devices collaboratively work towards completing tasks?* was answered in chapter 4 and 5. This was done by splitting a task when reaching a specific threshold and then asking other nodes in the network to help compute the generated task.

The answer to the third research question *How effective is the proposed novel architecture model in terms of task completion time, data size, and scalability?* was covered in chapter 6. The results concluded for the completion time for a task that the one-node testbed performed worse than the two and three-node testbeds. While the two and three-node testbeds performed likewise for when the threshold for helping others was set to zero. The three-node testbed did however perform overall worse for the other threshold configurations in comparison to the two-node testbed. The data size did not seem to have a major impact on the results. The scalability did not have a large impact on the result for the zero threshold for helping others but could potentially have been impacted on the other threshold configurations, especially for the three-node testbed.

The answers to the fourth research question *What are the benefits and drawbacks of the proposed novel architecture model?* were provided in chapters 4, 6, and 7. The benefits of the novel architecture model are the proven possibility of splitting tasks and the collaborative aspects that comes with it. The drawbacks however are the limitation to the size of data that can be transferred in the network which is partly an effect of the hardware which runs the model as well as the internet connection that the testbed runs on.

The work of this thesis contributes to the exploration of fog computing as an option to cloud computing and how it can be used for future IoT applications. The exploration was done by a literature study together

with designing, implementing, and evaluating a novel architecture model. The results show that a testbed built with RPi's is capable of collaborating and communicating to complete a task that has been distributed among multiple devices. The fog testbeds for one, two, and three devices do not computationally perform better than the reference testbed. However, increasing the number of fog devices and changing the threshold for when a device needs help with a task and keeping the threshold for helping others at zero could possibly provide better computational power and by doing so could potentially reach the power of the reference model or the cloud.

8.1 Future Work

There are many possibilities regarding future work expanding the work of this thesis. Three of these are presented below.

8.1.1 Connection and communication

There are a few changes that could be made for the implementation in future work. One is either running the testbeds on a more isolated network or by using ethernet-cables for the internet connection. A further development area is to split the generated tasks more dynamically for when help is needed based on the current workloads of fog nodes. This would however require the fog node that needs help to gather information about other nodes in the network to then distribute the task accordingly. This could require the number of fog nodes operating in a network to be relatively small due to scalability aspects since multiple simultaneous connection might need to be maintained throughout the lifespan of the network for it to properly work.

A careful evaluation of the scalability would need to be performed for this type of solution since multiple maintained connections might overwhelm the nodes in the network. Another evaluation area that would be interesting to investigate for this could be utilization factor of the network.

8.1.2 Cloud testbed

Implementation of a cloud testbed instead of the reference testbed implemented in this thesis could provide more insight to different aspects of using the cloud. This would require an investigation of different cloud service providers, most interesting is the prominent ones such as Microsoft, Google, or Amazon. There are two surveys that have

investigated different cloud service providers with focus on IoT and summarized different aspects, the first one done by Pflanzner and Kertész [39] and the second one done by Ray [40]. These two surveys could be a good starting point for this type of future work.

What would be interesting to evaluate here is the cost or different pricing models for this type of test bed in comparison to the fog testbeds. To appropriately evaluate performance aspects between the fog and cloud testbeds would require the fog testbed to include more nodes. Security is also something that could be looked in to for comparison reasons.

8.1.3 Kubernetes

A further research possibility that would be interesting to investigate is Kubernetes cluster-technology. Kubernetes is an orchestration software that helps run containerized applications containing multiple different components [41]. Each fog node would then be represented by a container. If the workload of the cluster of containers would become overwhelmed with tasks, Kubernetes would then be able to add another fog node to help with the tasks.

As an introduction to this type of technology and how to apply this in a suitable setting, one could investigate the research done by Kristiani *et al*. [42]. They use Kubernetes and OpenStack to implement a cloud-edge testbed for monitoring air quality.

References

[1] I. Yaqoob, E. Ahmed, I. A. T. Hashem, A. I. A. Ahmed, A. Gani, M. Imran and M. Guizani, "Internet of Things Architecture: Recent Advances, Taxonomy, Requirements, and Open Challenges," *IEEE Wireless Communications,* pp. 10-16, 1 June 2017.

[2] M. M. Sadeeq, N. M. Abdulkareem, S. R. M. Zeebaree, D. M. Ahmed, A. S. Sami and R. R. Zebari, "IoT and Cloud computing issues, challenges and opportunities: A review," *Qubahan Academic Journal,* vol. 1, p. 1–7, 2021.

[3] P. Corcoran, "The Internet of Things: Why now, and what's next?," *IEEE Consumer Electronics Magazine,* vol. 5, pp. 63-68, 2016.

[4] C. J. Kale and T. J. Socolofsky, *TCP/IP tutorial,* RFC Editor, 1991.

[5] L. Atzori, A. Iera and G. Morabito, "The Internet of Things: A survey," *Computer Networks,* vol. 54, pp. 2787-2805, 2010.

[6] T. Liu and D. Lu, "The application and development of IOT," in *2012 International Symposium on Information Technologies in Medicine and Education*, 2012.

[7] M. Hasan, "IoT Analytics," 18 May 2022. [Online]. Available: https://iot-analytics.com/number-connected-iot-devices/. [Accessed 18 May 2022].

[8] S. Kaur and I. Singh, "A survey report on Internet of Things applications," *International Journal of Computer Science Trends and Technology,* vol. 4, p. 330–335, 2016.

[9] J. H. Nord, A. Koohang and J. Paliszkiewicz, "The Internet of Things: Review and theoretical framework," *Expert Systems with Applications,* vol. 133, p. 97–108, 2019.

[10] P. Bellavista, J. Berrocal, A. Corradi, S. K. Das, L. Foschini and A. Zanni, "A survey on fog computing for the Internet of Things," *Pervasive and Mobile Computing,* vol. 52, pp. 71-99, 2019.

[11] J. Pan and J. McElhannon, "Future Edge Cloud and Edge Computing for Internet of Things Applications," *IEEE Internet of Things Journal,* vol. 5, pp. 439-449, 2018.

[12] S. Yi, C. Li and Q. Li, "A Survey of Fog Computing: Concepts, Applications and Issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*, New York, NY, USA, 2015.

[13] D. Vasconcelos, R. Andrade, V. Severino, J. De and Souza, "Cloud, Fog, or Mist in IoT? That Is the Question," *ACM Transactions on Internet Technology,* vol. 19, p. 25, March 2019.

[14] P. Galambos, "Cloud, Fog, and Mist Computing: Advanced Robot Applications," *IEEE Systems, Man, and Cybernetics Magazine,* vol. 6, pp. 41-45, 2020.

[15] M. Van Steen and A. S. Tanenbaum, Distributed systems, Maarten van Steen Leiden, The Netherlands, 2017.

[16] M. Al-Khafajiy, T. Baker, H. Al-Libawy, Z. Maamar, M. Aloqaily and Y. Jararweh, "Improving fog computing performance via fog-2-fog collaboration," *Future Generation Computer Systems,* vol. 100, p. 266–280, 2019.

[17] H. Shah-Mansouri and V. W. S. Wong, "Hierarchical fog-cloud computing for IoT systems: A computation offloading game," *IEEE Internet of Things Journal,* vol. 5, p. 3246–3257, 2018.

[18] E. Maskin, "The theory of implementation in Nash equilibrium: A survey," 1983.

[19] S. Shen, Y. Han, X. Wang and Y. Wang, "Computation offloading with multiple agents in edge-computing–supported IoT," *ACM Transactions on Sensor Networks (TOSN),* vol. 16, p. 1–27, 2019.

[20] K. Peffers, T. Tuunanen, M. A. Rothenberger and S. Chatterjee, "A design science research methodology for information systems research," *Journal of management information systems,* vol. 24, p. 45–77, 2007.

[21] P. Varshney and Y. Simmhan, "Demystifying fog computing: Characterizing architectures, applications and abstractions," in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, 2017.

[22] R. Mahmud, R. Kotagiri and R. Buyya, "Fog computing: A taxonomy, survey and future directions," in *Internet of everything*, Springer, 2018, p. 103–130.

[23] R. Mahmud, K. Ramamohanarao and R. Buyya, "Application management in fog computing environments: A taxonomy, review and future directions," *ACM Computing Surveys (CSUR),* vol. 53, p. 1–43, 2020.

[24] V. Kesri, V. Kesri and P. K. Pattnaik, "An unique solution for N queen problem," *International Journal of Computer Applications,* vol. 43, p. 1–6, 2012.

[25] P. Prajapati, N. Bhatt and N. Bhatt, "Performance comparison of different sorting algorithms," *vol. VI, no. Vi,* p. 39–41, 2017.

[26] R. Szeliski, Computer Vision: Algorithms and Applications, Springer, 2021.

[27] "Raspberry Pi," [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/. [Accessed 15 May 2022].

[28] "Alternative Python Implementations," [Online]. Available: https://www.python.org/download/alternatives/. [Accessed 20 May 2022].

[29] "Python Glossary," [Online]. Available: https://docs.python.org/3/glossary.html#term-global-interpreter-lock. [Accessed 21 May 2022].

[30] "Python Threading," [Online]. Available: https://docs.python.org/3/library/threading.html. [Accessed 20 May 2022].

[31] "Numpy Random Sampling," [Online]. Available: https://numpy.org/doc/stable/reference/random/index.html. [Accessed 22 May 2022].

[32] "Numpy PCG," [Online]. Available: https://numpy.org/doc/stable/reference/random/bit_generators/pcg64.html#numpy.random.PCG64. [Accessed 15 May 2022].

[33] "Python Time access and conversions," [Online]. Available: https://docs.python.org/3/library/time.html. [Accessed 11 May 2022].

[34] "Numpy Random Shuffle," [Online]. Available: https://numpy.org/doc/stable/reference/random/generated/numpy.random.shuffle.html. [Accessed 11 May 2022].

[35] "PennState Eberly College of Science," [Online]. Available: https://online.stat.psu.edu/stat200/lesson/3/3.2. [Accessed 10 May 2022].

[36] "Python Multiprocessing," [Online]. Available: https://docs.python.org/3/library/multiprocessing.html. [Accessed 16 May 2022].

[37] E. Al Nuaimi, H. Al Neyadi, N. Mohamed and J. Al-Jaroodi, "Applications of big data to smart cities," *Journal of Internet Services and Applications,* vol. 6, p. 1–15, 2015.

[38] G. Corotinschi and V. G. Găitan, "The development of IoT applications using old hardware equipment and virtual TEDS," in *2016 International Conference on Development and Application Systems (DAS)*, 2016.

[39] T. Pflanzner and A. Kertész, "A survey of IoT cloud providers," in *2016 39th International convention on information and communication technology, electronics and microelectronics (MIPRO)*, 2016.

[40] P. P. Ray, "A survey of IoT cloud platforms," *Future Computing and Informatics Journal,* vol. 1, p. 35–46, 2016.

[41] "Azure Microsoft," [Online]. Available: https://azure.microsoft.com/en-us/topic/what-is-kubernetes/#beyond-kubernetes. [Accessed 14 May 2022].

[42] E. Kristiani, C.-T. Yang, C.-Y. Huang, Y.-T. Wang and P.-C. Ko, "The implementation of a cloud-edge computing architecture using OpenStack and Kubernetes for air quality monitoring application," *Mobile Networks and Applications,* vol. 26, p. 1070–1092, 2021.

Appendix A: Related efforts

Characteristics	Cloudlet	Fog Computing	ETSI MEC Initiative	CORD	NEBULA	FemtoCloud	Cloud offloading and Foraging	HomeCloud
Advocates and Sponsors	Academia	Industry; multiple vendors	Industry; Wireless telecom providers	Industry; Wireline service providers	Academia	Academia	Academia	Academia
Key design goals	Mobile computing and cloud computing convergence	Reduce data movement across networks; Bring all "networks" and all "things" in.	Transition mobile cellular networks toward 5G with edge cloud capabilities	Wireline telecom access networks	Location and context-aware edge cloud for data-intensive computing	Share idle resources among mobile devices	Mobile-edge offloading and foraging	Automated, open, and portable new IoT app delivery
Key design features	1. 3-tier cloud structure; 2. "datacenter in a box"; 3. Dynamic VM synthesis	Near-edge servers; Control plane and data plane separation	Radio Access Network (RAN) open to third-parties; MEC servers in flexible locations (eNodeB, RNC, etc)	Software and hardware separation in Central Offices (COs); using open source software on "whiteboxes"	Allows edge volunteers to contribute for distributed MapReduce apps	A group of mobile devices function as a cluster	Program partitioning and coordination between mobile and edge	NFV+SDN; Automated orchestration; multiple apps on same infrastructure; portability
Key applications	crowd-sourced video surveillance and cognitive assistance	Potentially all kinds of IoE, connected vehicles, WSN, CPS, smart buildings, etc.	Mobile cellular apps; Location tracking; Radio aware video optimization, etc.	Virtualize COs and deliver end-to-end SDN/NFV/Cloud solutions	Distributed data-intensive applications such as MapReduce	Mobile device apps in Coffee shops, classrooms, theaters, etc	Mobile programs partitioned to allow offloading parts to servers.	Future IoT apps, smart home, smart energy, VR, AR, etc
Infrastructure sever support	Yes, edge "boxes"	Yes, edge servers	Yes, at base stations, etc	Yes, at central offices (COs)	No dedicated edge servers	No, but with a task controller	Yes, at edge servers	Yes, at edge servers
Virtualization at the edge	Yes, extends OpenStack	Not specified	Yes	Yes	Not specified	No	No, but by program partitioning	Yes
SDN at the edge	Not specified	Not specified	Not specified	Yes	Not specified	No	No	Yes
Mobility support	Possibly VM migration	Not specified	Yes, but depends on the specific apps	Not specified	Dynamic and mobile volunteers at edges	Not specified	Process migration and code partitioning	Support high level mobility (user, data, VMs)
Application portability	Not specified	Not specified	Possibly yes, mechanisms unclear yet	Not specified	Not specified	Not specified	Applications not platform-independent	Yes, as principal designs goals
Security considerations	Reduce data transmission risks; edge apps failover	Reduce data attacks enroute; edge tolerant to network failure	Need to fulfill 3GPP security and provide secure sandbox for apps	Telecom level security requirements	Tolerant to compute node failure and data node failure	Difficult due to mobile devices' high volatility, dynamicity, and instability	Software security concerns related to code partitioning and coordination	Reduce data transmission risks; Reduce data attacks enroute; apps isolation and dynamic start/end