

University of Fort Hare

Together in Excellence

Resource Allocation Framework in Fog Computing for the Internet of Things Environments

A Thesis Submitted in Fulfilment of the
Requirements for the Degree
of
Doctor of Philosophy in Computer Science

Submitted by

William Tichaona Vambe

Student Number: 201515191

Supervised by

Prof Khulumani Sibanda

[January 2020]

Abstract

Fog computing plays a pivotal role in the Internet of Things (IoT) ecosystem because of its ability to support delay-sensitive tasks, bringing resources from cloud servers closer to the “ground” and support IoT devices that are resource-constrained. Although fog computing offers some benefits such as quick response to requests, geo-distributed data processing and data processing in the proximity of the IoT devices, the exponential increase of IoT devices and large volumes of data being generated has led to a new set of challenges. One such problem is the allocation of resources to IoT tasks to match their computational needs and quality of service (QoS) requirements, whilst meeting both task deadlines and user expectations. Most proposed solutions in existing works suggest task offloading mechanisms where IoT devices would offload their tasks randomly to the fog layer or cloud layer. This helps in minimizing the communication delay; however, most tasks would end up missing their deadlines as many delays are experienced during offloading. This study proposes and introduces a Resource Allocation Scheduler (RAS) at the IoT-Fog gateway, whose goal is to decide where and when a task is to be offloaded, either to the fog layer, or the cloud layer based on their priority needs, computational needs and QoS requirements. The aim directly places work within the communication networks domain, in the transport layer of the Open Systems Interconnection (OSI) model. As such, this study follows the four phases of the top-down approach because of its reusability characteristics. To validate and test the efficiency and effectiveness of the RAS, the fog framework was implemented and evaluated in a simulated smart home setup. The essential metrics that were used to check if round-trip time was minimized are the queuing time, offloading time and throughput for QoS. The results showed that the RAS helps to reduce the round-trip time, increases throughput and leads to improved QoS. Furthermore, the approach addressed the starvation problem, a phenomenon that tends to affect low priority tasks. Most importantly, the results provides evidence that if resource allocation and assignment are appropriately done, round-trip time can be reduced and QoS can be improved in fog computing. The significant contribution of this research is the novel framework which minimizes round-trip time, addresses the starvation problem and improves QoS. Moreover, a literature reviewed paper which was regarded by reviewers as the first, as far as QoS in fog computing is concerned was produced.

Keywords: *Internet of Things, Cloud Computing, Fog Computing, Quality of Service, Round-trip Time, Resource Allocation.*

Publications from this Thesis

Published

2019. “A Review of Quality of Service in Fog Computing for the Internet of Things”. ***International Journal for Fog Computing, 3 (1)***. <https://www.igi-global.com/article/a-review-of-quality-of-service-in-fog-computing-for-the-internet-of-things/245708>, **DOI:** 10.4018/IJFC.2020010102
(Based on Chapter Two)


Accepted for publication

2020. “A Fog Computing Framework for Quality of Service Optimisation in the Internet of Things (IoT) Ecosystem” submitted for a conference <https://www.spu.ac.za/index.php/ieee-imatec-2020/> and paper will be published in the IEEE digital library.

Declaration

I **William Tichaona Vambe** the undersigned, student number **201515191**, do hereby declare that the thesis titled “**Resource allocation framework in fog computing for the Internet of Things environments**” for **Doctor of Philosophy in Computer Science** is my original work in design and execution. The work has not been submitted or presented at any other University for a similar or any other degree award. All reference materials used have been duly acknowledged.

I hereby further declare that I am fully aware of the University of Fort Hare’s policy on plagiarism and research ethics, and I have taken every necessary measure to comply with the regulations.

Signature: 

Date: *31/01/2020*

William Tichaona Vambe

Acknowledgments

I would like to thank my supervisors, Prof. Khulumani Sibanda, from the Department of Computer Science at the University Fort Hare, Dr Chii Chang from the University of Melbourne in Australia for the guidance and feedback throughout the study. Further, I would also want to thank Prof Satish Srirama (head of mobile and cloud laboratories) and the University of Tartu (Estonia), for giving me the Dora Plus grant. The grant helped me a lot in my research when I spent time at their university as a visiting PhD student.

I am profoundly indebted to Christine Munemo, Victor Vambe, Elphigio P. Vambe, Washington Vambe, Berrelyn Rosina (Mabuku) Vambe and Sarudzai (Chipfumbu) Vambe for the unconditional love and unwavering support during this study. Thank you. Also, I would like to thank Patricia Vambe, Clarity (Hutete) Vambe, my brothers Byron, Leeroy, Simbarashe and Wilfred, my sister Evangelista Vambe, uncles, aunties and my friends Tineyi Herbert Pindura, Dr Tafadzwa Maramura and Dr E.Chindenga. Through thick and thin, you have always been supportive of my life and education.

The Head of Department Mr S. Dyakalashé and Mr M.S Scott of the Computer Science Department at the University of Fort Hare and the Institute of Computer Science at the University of Tartu is also hereby acknowledged. Lastly, I would like to acknowledge the financial assistance from the University of Fort Hare Govan Mbeki Research & Development Centre (GMRDC).

The opinions and views expressed are of the author and do not reflect those of the University of Fort Hare.

Dedication

In memory of:

My first teachers who happen to be my grandfather and grandmother (Mr. and Mrs.) Patricio Muvirimi Makhosa and Esnarth (Muza)

Makhosa.

My hero, my advisor, pillar of strength, my grandfather Mr William M'haiwa Vambe, who believed in me, sacrificed a lot and encouraged me in this academic journey and life in general.

My three children (2 boys and a girl) I lost as stillbirths from 2018 to 2020 (My PhD years)

Although you are far away now, I forever thank you, and I am grateful for all the love you gave me. We meet to part and part to meet. May all your souls. I dedicate this to you.

Rest in Peace as I continue being your ambassador.

Table of Contents

Abstract.....	i
Publications from this Thesis	ii
Declaration	iii
Acknowledgments.....	iv
Dedication.....	v
List of Tables	x
List of Figures	xi
Acronyms	xii
Chapter One: Introduction	1
1.1 Introduction.....	1
1.2 Background of the Study	1
1.3 Overview of Related Work.....	3
1.4 Problem Statement.....	5
1.5 Research Aim	5
1.6 Research Objectives	5
1.7 Research Questions	6
1.8 Overview of Research Methodology.....	6
1.9 Contributions of the Study	6
1.10 Limitations of this Study.....	7
1.11 The Structure of the Thesis	7
1.12 Conclusion.....	8
Chapter Two: Literature Review	9
2.1 Introduction.....	9
2.2 Internet of Things.....	9
2.2.1 IoT Characteristics	10
2.3 Cloud Computing.....	13

2.3.1	Cloud Computing Characteristics.....	13
2.3.2	Service Models	15
2.3.3	Deployment Models	15
2.3.4	Virtualization	16
2.3.5	IoT-Cloud Architecture	18
2.4	Fog Computing	20
2.4.1	Fog Computing Characteristics.....	20
2.4.2	IoT-Fog-Cloud Architecture.....	22
2.5	Application Area : Smart Home	24
2.6	Quality of Service	24
2.7	Related Work.....	26
2.7.1	Resource Provisioning and Scheduling for Fog Computing Environment.....	26
2.7.2	Scheduling techniques	33
2.7.3	Round-trip Time (Latency)	35
2.7.4	Software Frameworks	36
2.8	Research Gap in the Current Fog Computing Research Provisioning Strategies.....	37
2.9	Conclusion.....	39
	Chapter Three: Research Design and Methodology.....	40
3.1	Introduction.....	40
3.2	Research Process Design	40
3.2.1	Requirement Analysis	42
3.2.2	Logical Network Design	46
3.2.3	Physical Network Design	49
3.2.4	Testing and Evaluation.....	49
3.3	Conclusion.....	51
	Chapter Four: Framework Design and Implementation	52
4.1	Introduction.....	52
4.2	Framework Design	52
4.2.1	Components of the Framework.....	53
4.2.1.1	Edge Layer (IoT devices)	53

4.2.1.2	IoT-Fog Gateway (Resource Allocation Scheduler).....	54
4.2.1.3	Fog Layer	61
4.2.1.4	Cloud Layer	63
4.2.2	Aspects Expected to be Supported with the Framework.....	64
4.2.3	Workflows	64
4.2.4	<i>Application Programming Interfaces (APIs) Endpoints</i>	67
4.3	Implementation	70
4.3.1	Testing of an IoT Service.	70
4.3.2	IoT Application Execution.....	73
4.3.3	Testing of the Resource Allocation Scheduler	74
4.3.4	Evaluation Round-Trip Time	75
4.3.4.1	Queueing Time	76
4.3.4.2	Offloading Time Evaluation	76
4.4	Conclusion.....	78
	Chapter Five: Results and Discussions	79
5.1	Introduction.....	79
5.2	RAS Performance Evaluation Setup.....	79
5.2.1	Simulated Environment Description	79
5.3	Performance Evaluation.	80
5.3.1	Results.....	80
5.3.2	Findings	91
5.3.3	Result Findings Analysis	93
5.3.4	Discussion of Results Findings	95
5.4	Conclusion.....	97
	Chapter Six: Conclusion and Future Work	98
6.1	Introduction.....	98
6.2	Summary of the Study	98
6.3	Research Objectives, and Where Addressed	100
6.4	Contributions of this Thesis	102
6.5	Future Work.....	103

6.6	Conclusions.....	104
	References	105

List of Tables

Table 3-1: <i>Top-Down Research Methodology Phases</i>	41
Table 4-1: <i>Register Service Endpoint</i>	68
Table 4-2: <i>Send Task Request Endpoint</i>	68
Table 4-3: <i>Get Resource Utilization Endpoint</i>	69
Table 4-4: <i>Resource Allocation Scheduler Propagator Endpoint</i>	69
Table 4-5: <i>Cloud Fog Middleware Propagator Endpoint</i>	69
Table 5-1: <i>Simulation Parameters</i>	80
Table 6-1: <i>Research Objectives and Where they were Addressed</i>	101

List of Figures

Figure 2-1: <i>Cloud Computing Overview</i>	13
Figure 2-2: <i>IoT-Cloud Architecture</i>	18
Figure 2-3: <i>IoT-Fog-Cloud Architecture</i>	22
Figure 3-1: <i>Top-Down Research Methodology phases</i>	42
Figure 3-2: <i>Network Design Topology</i>	47
Figure 4-1: <i>Cross-Sectional Design of the Fog Computing Framework</i>	53
Figure 4-2: <i>Decision Making Flowchart</i>	54
Figure 4-3: <i>First Fit Algorithm in RAS for Resource Provisioning</i>	55
Figure 4-4: <i>Priority queueing model for IoT gateway</i>	57
Figure 4-5: <i>Reasoner, Redis FCN, Redis Shared Deployed on the Raspberry Pi</i>	61
Figure 4-6: <i>Fog Node Deployment on the Raspberry Pi</i>	63
Figure 4-7: <i>Pairing of Fog Nodes, RSA, IoT Devices and Service Deployment</i>	65
Figure 4-8: <i>RAS Assignment of Task and Processing</i>	67
Figure 4-9: <i>Node-RED flowchart</i>	73
Figure 5-1: <i>Queueing time for high priority tasks, low priority tasks, and no priority tasks</i>	81
Figure 5-2: <i>Offloading time for High priority tasks, Low priority tasks, and No priority tasks</i>	83
Figure 5-3: <i>Illustration of task data arrival, task execution and task offloading</i>	85
Figure 5-4: <i>Performance-based on average queueing time</i>	88
Figure 5-5: <i>Performance-based on average offloading time</i>	89
Figure 5-6: <i>Performance-based on average percentage number of tasks satisfying delay deadlines</i>	90
Figure 5-7: <i>Performance-based on average throughput</i>	91

Acronyms

API	Application Programming Interface
AV	Availability
BPDOC	Building Process Documentation Cloud
CaaS	Control as a Service
4IR	Fourth Industrial Revolution
FiWi	Fiber-Wireless
FRED	Front End for Node-RED
FRL	Fog Reallocation
FRR	Fog Resource Reservation
FME	Follow me Edge
FSPP	Fog Service Placement Problem
HTTP	Hypertext Transfer Protocol
GUI	Graphic User Interface
IaaS	Infrastructure-as-a-Service
IoT	Internet of Things
IP	Internet Protocol
IoV	Internet of Vehicles
MEC	mobile edge computing
MPSO-CO	Modified Constrained Optimization Particle Swarm Optimization
MSA	Micro Services Architecture
NFC	Near Field Communication
NIST	National Institute of Standards and Technology
OS	Operating System
PaaS	Platform-as-a-Service
PIOTS	Pattern-Identified Online Scheduling Task
PO	Priority
PTPN	Priced Timed Petri Nets
QoS	Quality of Service
QoE	Quality of Experience
RAS	Resource Allocation Scheduler
RFI	Radio-Frequency Identification.
VM	Virtual Machine.
Wi-Fi	Wireless Fidelity
WSN	Wireless Sensor Networks
SaaS	Software-as-a-Service
TD	Transit Delay
TP	Throughput

Chapter One: Introduction

1.1 Introduction

This chapter provides an overview of this study and establishes the research niche. The chapter is structured as follows: **Section 1.2** gives the background of the study, followed by **Section 1.3**, which provides the synopsis with an overview of related work. The problem statement is then presented in **Section 1.4**. The research aims and research questions are given in **Section 1.5** and **Section 1.7**, respectively. **Section 1.6** provides the research objectives. The overview of the research methodology is presented in **Section 1.8**. **Section 1.9** gives the contribution of the study. The limitations of this study are then presented in **Section 1.10**. The chapter is concluded by providing the structure of the whole thesis in **Section 1.11**.

1.2 Background of the Study

With the advent of the Internet of Things (IoT), which is creating a “smart world” and bringing about automation in many application areas, many computing elements need various modifications to support the IoT devices that are at the center of the automation world. Such changes should help the IoT devices, which are resource-constrained, while keeping in mind that latency has to be minimized and Quality of Service (QoS) has to be improved. It is important to note that for successful adoption of IoT, it should be associated with a wide variety of other technologies such as cloud computing and fog computing.

Cloud computing was introduced to support IoT devices in terms of resources (Chen *et al.*, 2017). Although cloud computing as a concept dates back to the 1990s, the term cloud computing was first used in 2006, on the 9th of August by Eric Schmidt, Chairman and CEO of Google at the Search Engine Strategies Conference (Google Press, 2006). Since then, cloud computing has been widely adopted in many businesses for backup, file storage, cost-cutting in terms of infrastructure, development and testing as well as investment by cloud providers. Cloud computing takes a central role to support emerging IoT technologies which are resource constrained. The cloud computing has become vital in supporting the interactions between IoT networks. However, the exponential growth of the number of connected sensors is becoming a challenge to the cloud architecture. This is because cloud computing is a centralised approach

which makes it difficult to service geo-distributed IoT devices. The geographical distance between IoT devices and cloud servers seriously affects how the two communicate, leading to undesirable latency challenges. Secondly, it becomes costly to send IoT generated tasks to and from the cloud servers as more bandwidth is needed during the transmission.

Due to the above-mentioned challenges, fog computing was introduced by Cisco in 2012, not as a substitute for cloud computing, but to complement cloud computing (Bonomi *et al.*, 2012). OpenFog Consortium Architecture Working Group defined fog computing as “a *system-level horizontal architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Things*” (OpenFog Consortium Architecture Working Group, 2017). It is made up of both wired and wireless granular collection endpoints, which include switching equipment, routers that act as gateways and customer premise equipment (CPE). Fog computing has become a preferred choice because of its ability to deliver services faster, and its ability to offer location awareness. It is worth to reiterate that fog computing technology is not a replacement of cloud computing but complements it by bringing the “cloud resources closer to the ground” where IoT devices reside (Chang *et al.*, 2017).

As evidenced in the detailed survey done by Vambe *et al.*, (2020), several studies have focused on addressing various fog computing issues. One of the topics that is drawing much attention is how communication and computing resources can be allocated and assigned based on tasks, requirements and priorities. The existing solutions, as informed in literature, indicate that resources are assigned/ offloaded based on a first come first serve basis without considering task status (whether a task is time-sensitive or not) and in most cases, task deadlines (Vambe *et al.*, 2020). Some existing works focus only on the reduction of communication delay. Despite many efforts being done to reduce communication delay, this study discovered that in many proposed solutions, most time-sensitive tasks fail to meet their deadlines. This situation can severely affect automation. The starvation problem is another challenge that is receiving much attention from various researchers. Another open challenge in fog computing is to find an effective and efficient resource allocation and assignment mechanism that meets the needs of both time-sensitive tasks and those that are not time-sensitive while meeting tasks deadlines. Hence, the primary goal of this study was to introduce a Resource Allocation Scheduler (RAS) which reduces round-trip time for time sensitive tasks and also helps to solve the starvation

problem that affects IoT tasks that are not time sensitive. To achieve this goal, this study proposes and implements a Resource Allocation Scheduler (RAS) in the fog computing framework. RAS was introduced at the IoT-Fog gateways whose responsibility was to allocate and assign tasks generated by IoT devices to either fog layer or cloud layer based on the task's computational needs and priority.

1.3 Overview of Related Work

Literature shows that several researchers have used fog computing to minimise latency and improve QoS in existing systems. Such work includes the work of Kochovski and Stankovski, (2018), who applied Jitsi-meet and building process documentation cloud (BPDOC) applications to promote orchestration of services to address the QoS hindrance problem in the smart construction domain. Alsaffar *et al.*, (2017) discussed the issue of service placement in a home setup domain using a resource allocation algorithm to optimize data distribution and resource allocation. A two computational algorithm with low delay and reduced complexity that uses the principle of computation offloading in a mobile domain was used by Liu *et al.*, (2017) to address service migration mobility. A Follow me Edge (FME) concept was used by Taleb *et al.*, (2017) in a smart city domain to achieve efficient resource deployment as a way to address the service migration problem. Modified Constrained Optimization particle swarm optimization (MPSO-CO) was applied by He *et al.*, (2016) on the Internet of Vehicles (IoV) domain to address load balancing challenges. Sampei, (2017) suggested combining network slicing, network softwarization, and mobile edge computing (MEC) to address challenges faced when expanding cellular service to achieve efficient network flexibility. Power minimization resource algorithms MC-RAN was used by Wang and Yang, (2017) in the mobile application domain to assist resource hungry and computational limited devices so that they will be able to dynamically compute resource allocation. Li *et al.*, (2018) devised a method of resource estimation. It was based on QoS in Edge computing, which used multi-attribute QoS resource matching algorithm and regression Markov prediction method to forecast available resources, select the suitable resource to meet the needs of users. Thus, reducing unnecessary competition for the resource, which improves QoS. Souza *et al.*, (2017) argued that QoS is not only affected by data transmission factors but also processing delays in fog nodes. To address the end-to-end delay in fog computing, Souza *et al.*, (2017) introduced a service-oriented control that would allow control as a service (CaaS) in the fog to cloud topology. Fog Resource Reservation (FRR) and

Fog Reallocation (FRL) strategies were introduced by Li *et al.*, (2017) in fog computing after the realization that fog nodes have limited resources when it comes to processing power. As such, they can quickly become overloaded when large amounts of users' requests arrive during peak hours, resulting in processing delays that will in-turn affect QoS. Xiao and Krunz, (2017), offload forwarding strategy was introduced to address service migration challenges in fog computing networks. A fog node would either not offload or offload and forward part or its entire load to be processed by other local fog nodes that are idle and have better computational power than it has. Task distribution algorithm, which was based on initialization, relaxation, rounding, and validation, was introduced by Song *et al.*, (2017) to address the service migration problem in fog computing that affected QoS. Skarlat *et al.*, (2017) designed a novel Fog Service Placement Problem (FSPP) method that would facilitate optimal sharing of resources.

Although several related works tried to address QoS issues in IoT ecosystems, the challenge that still remains is the allocation and offloading of the tasks that are produced by the IoT devices to the resources that suit their computational needs and fulfil their QoS requirements. Moreover, to the resources that suit their deadline needs while minimizing round-trip time (Vambe *et al.*, 2020). Some researchers have made efforts to solve this challenge. For example, Ko *et al.*, (2017) and Mukherjee *et al.*, (2019) have investigated and suggested ways on how to address the problem of task allocation and offloading. The latest research by Yang *et al.*, (2019) and Wang *et al.*, (2019) suggested offloading tasks to nearby fog nodes or cloud servers.

It is worth pointing out that all these works have one thing in common. The decision is made in the fog nodes to either process the whole tasks, part of the task or offload to the next fog node. This clearly shows that when tasks are sent to the fog layer, deadline requirements of tasks are not considered. Deadline requirements play a pivotal role when considering time-sensitive tasks as they require to be processed at a specific time frame. Failure to meet deadlines implies that if the outcome of the task comes after the stipulated time, it becomes useless. This can be detrimental in critical applications like medical health applications. Hence this study proposed a solution that would help tasks meet their deadlines by minimizing round-trip delays and addressing the starvation problem. The solution of this study introduced a Resource Allocation Scheduler (RAS) in the IoT-Fog gateways that is responsible for resource allocation, giving high priority to time-sensitive tasks. The RAS considers task deadlines, resource constraints and promote minimized latency. This research is of paramount importance as several application

areas such as smart health, smart city, smart grids would benefit from the findings of this research.

1.4 Problem Statement

The exponential increase of IoT devices and large volumes of data generated by IoT devices has led to severe challenges in the allocation of resources to IoT tasks in order to match their computational needs and achieve QoS requirements while meeting task deadlines, at the same time avoiding the starvation problem. Trying to improve quality of service, the use of fog nodes has been adopted. However, fog nodes are resource constrained. Overloading them leads to negative impact in terms of turnaround time of network packets. This calls for techniques that can reduce turn around time of network packets that are sent to fog nodes, thereby improving QoS in the internet of things environments.

1.5 Research Aim

This study aimed to design and implement a resource allocation scheduler in fog computing framework for the Internet of Things (IoT) environment. To pursue this aim, the study used the following research objectives and research questions as a guide:

1.6 Research Objectives

1. To identify the key challenges of data communication and computer resources allocation in an IoT environment.
2. To determine how data communication and computer resources are assigned and allocated based on tasks requirements and their priorities in IoT environments.
3. To identify a suitable methodology for building a resource allocation scheduler in fog computing framework for IoT environments.
4. To build a resource allocation scheduler framework in fog computing for IoT environments.
5. To test and evaluate the effectiveness of the proposed RAS in allocating resources in fog computing framework.

1.7 Research Questions

1. What are the key challenges in communication and computer resources allocation in an IoT environment?
2. How are communication and computing resources allocated and assigned among the IoT devices based on tasks, requirements and priorities?
3. Which approaches can be used to build a resource allocation scheduler in fog computing framework for IoT environment?
4. Can a resource allocation scheduler in fog computing framework which is based on tasks requirements and priorities be successfully developed?
5. What is the performance of the RAS in fog computing framework?

1.8 Overview of Research Methodology

The development of the RAS in fog computing framework was informed by systematic scrutiny of existing approaches as a way to discover the weakness of the existing frameworks. The aim directly put this work within the communication networks domain, in the transport layer of the Open Systems Interconnection (OSI) model. As such, this study followed the four phases of the top-down approach because of its reusability characteristics. The framework was then implemented and evaluated in a simulated smart home setup to validate and test the efficiency and effectiveness of the introduced RAS. The essential metrics that were used in the evaluation of this research were; a) queuing time, b) offloading time, which are factors of round-trip time that affect latency, and c) throughput, which is a parameter of QoS. More discussions on the study methodology used are in chapter three.

1.9 Contributions of the Study

There are two significant contributions which were made by this current study to the existing body of scientific knowledge. A review paper of QoS in fog computing was produced. To our understanding with the comments received from the reviewers, this was the first review paper which focuses on QoS in fog computing. The paper was published in the “*International Journal for Fog Computing*” (Vambe *et al.*, 2020). The paper provided a good starting point for discussion of QoS and how it can be improved in fog computing. Secondly, the novel resource allocation scheduler (RAS) brought about reduced queueing time and offloading time which

resulted in the minimization of round-trip time. Moreover, throughput was improved, which leads to improved QoS. The developed framework proved to be of more significant in smart environments such as smart home where it was tested. It showed that it could promote automation where time-sensitive applications can be served at real-time. This system can be used anywhere where round-trip time is to be minimized and QoS is to be improved.

In a nutshell, framework contribution serves as a foundation for further research in the field of fog computing. Moreover, the framework can be applied and tested for its benefits in application areas such as smart health where real-time responses are needed in real-time.

1.10 Limitations of this Study

This study focuses only on how to minimize round-trip time while considering queuing time and offloading time as an evaluation matrix. However, many other factors affect round-trip time and should be investigated in future work. Moreover, our work did not consider the security and privacy of data when making decisions in the RAS which we strongly believe they should be considered in future work. Considering security and privacy of data make users trust the use of IoT devices and fog layer devices which they will be using every day in a smart home setup.

1.11 The Structure of the Thesis

In **Chapter Two**, contextual knowledge which is the foundation to understand the research background is presented. The chapter gives an insight into the Internet of Things, cloud computing and fog computing technology. A critical analysis of work explicitly done which focused on resource provisioning and QoS in fog computing was presented. It is from this literature review where gaps in the existing framework as far as resource provisioning and improving QoS were identified. The gaps formed the basis of this thesis where there is a need to design and implement a resource allocation scheduler in fog computing framework for the IoT environment.

Within **Chapter Three: Research Design Methodology**, an overview of the top-down research methodology that was adopted and the reason why it was adopted in this research is presented. Moreover, a detailed explanation of what was done at each of the four stages of the top-down research methodologies to answer the research aim, research questions and objectives is presented.

Next, the most fundamental design of the whole framework and the implemented RAS in fog computing is presented in **Chapter Four: System Design and Implementation**. This chapter is of paramount importance as it defines the most fundamental design and functionalities of the framework, which are critical in the designing of the framework.

Chapter Five: Results and Discussions gives the empirical evaluation findings which are centered at queuing time, offloading time which are evaluation matrix for round-trip time. These results are presented in the form of graphs, followed by a critical analysis of each figure. A discussion of the findings of the developed RAS in fog computing as far as reducing round-trip time and improving QoS in relation to other works in literature is presented.

The thesis is concluded by presenting **Chapter Six: Conclusions and Future Work**. In this chapter, an overall word on the developed and implemented RAS in fog computing framework is presented. Acumens into the future work in the area of maintaining the existing standard or improving QoS in fog computing are also given.

1.12 Conclusion

This chapter started by giving a brief insight into the background of the study. In this background, a challenge of resource allocation in fog computing was identified, which was motivated to be the problem statement. The problem statement helped in the formulation of the research aim, research questions and research objectives. An overview on how the resech aim and questions were addressed is also highlighted. The contributions of this thesis to the body of knowledge was also presented. The chapter was concluded by giving an overall layout of the whole thesis.

Chapter Two: Literature Review

2.1 Introduction

This chapter answers the first research question, “What are the key challenges in communication and computer resources allocation in an IoT environment?”. Moreover, it answers the second research question, “How are communication and computing resources allocated and assigned among the IoT devices based on tasks, requirements and priorities?”. The chapter is structured as follows. The essential summary background of the Internet of Things looking at what it is, its characteristics and how it changed the existing information systems and applications is presented in **Section 2.2**. In **Section 2.3**, the synopsis of cloud computing that is its characteristics, service and deployment models that make it possible to be the central computational backbone of IoT is presented. Cloud computing challenges that make it not suitable to fully support IoT devices due to their characteristics are put across to justify why there was need of distributed computational paradigm called fog computing. In **Section 2.4**, the background of fog computing is highlighted. Fog computing characteristics that fulfil and comply with IoT needs are also presented. In **Section 2.5**, application areas of fog computing are highlighted. **Section 2.6** briefly describes the quality of service in the context of this research and highlights why it is important in fog computing. **Section 2.7**, presents related work as far as resource provisioning and scheduling in a fog computing environment is concerned. The chapter is concluded by giving an insight into the research gaps that still exist in the current fog computing resource allocation and scheduling strategies in **Section 2.8**. A summary of the whole chapter is given in **Section 2.9**.

2.2 Internet of Things

Presently, the Internet of Things (IoT) technology is trending all over the world in both academic and industries. The term IoT can be traced way back to the late 1990s when Kelvin Ashton introduced it and the vision being to connect intelligent “things” to the internet (Albishi *et al.*, 2017). Internet of Things (IoT) can be defined as “*a dynamic global information network consisting of internet-connected objects, such as radio-frequency identifications, sensors, and actuators, as well as other instruments and smart appliances that are becoming an integral component of the Internet*” (Perera *et*

al., 2014). Essentially, IoT is an interconnection of three “things” namely: a) Human beings/ animal to human beings/animal), b) Human beings/ animal to things/machine and, c) Things/machine to things/machine, interacting through the internet (Patel *et al.*, 2016). These “things” can connect to each other via Near Field Communication (NFC) (Dinh *et al.*, 2013), Radio-Frequency Identification (RFID) (Atzori *et al.*, 2010), Wireless Fidelity (Wi-Fi) (Atzori *et al.*, 2010), Bluetooth (Dilworth, 2012) and Wireless Sensor Networks (WSN) (Sensor, 2009). Near Field Communication and Radio-Frequency Identification use the concept of proximity to IoT devices to identify, authenticate and track them (Gubbi *et al.*, 2013).

Over the years, IoT has gained much attention and has become the centre of the fourth industrial revolution (4IR). Internet of Things can collect data from the physical environment, which upon processing the information generated can be used to generate insights for decision making in many application areas. Additionally, the data and events generated by IoT devices can be sent to the desired destination through the network, and upon further sophisticated analytics, the information created can be used to prompt for corresponding suitable actions. In short, IoT is a complex system that can create content, communicate, aggregate, analyse and act without explicit instructions (Perera *et al.*, 2014).

2.2.1 IoT Characteristics

The following are the fundamental characteristics of IoT, as highlighted in (Patel *et al.*, 2016) and (Vermesan and Friess, 2014), which makes it possible to do the aforementioned.

i. Intelligence

The fact that IoT is a combination of software and hardware, algorithms and computation makes it smart. Ambient intelligence in IoT improves its capabilities to enable the “things” to respond to a particular situation in an intelligent way and to assist them in performing specific tasks. For all the popularity of smart technology, IoT intelligence is regarded only as a means of interaction between devices. In contrast, the communication between users and devices is accomplished by input methods that are the standard and graphical user interface.

ii. Heterogeneity

Heterogeneity is one of the main characteristics of IoT. IoT devices are based on various hardware systems and networks and can communicate across different networks with other devices or service platforms. IoT architecture will allow direct access of heterogeneous networks to the network. Scalabilities, modularity, extensibility and interoperability are regarded as the key design requirements for heterogeneous things and their IoT environments.

iii. Interconnectivity

As far as IoT is concerned, and with the advancement of technology in the existence of global information and communication infrastructure, anything can be interconnected. Interconnectivity of these “things” is crucial because fundamental interactions at “things” level lead to mutual intelligence in the IoT network. In such a scenario, network accessibility and compatibility are enabled in the “things”. Through this connectivity, smart devices and apps network will be built forming new possibilities for the Internet of things.

iv. Dynamic changes

One significant role for IoT is to gather data from its surroundings, and this is achieved with the complex changes taking place around the devices. Devices state dynamically changes and the context of the devices which include temperature, position and speed. Aside from device status, the number of devices often dynamically changes with a person, location and time.

v. Sensing

Without sensors that detect or quantify any changes in the environment, IoT will not be possible to produce data that can communicate on their status or even interact with the environment. Sensing technologies provide the means to build capabilities that represent a true consciousness of the physical world and the people inside it. Information gathered from sensors is the physical world's analogue data, but it can provide our dynamic world's rich understanding.

vi. Enormous Scale

In 2011, Cisco Systems projected that 50 billion devices would be connected by the end of 2020 (Ericsson, 2011). This means the number of devices which need to be managed and communicate with each other will be much higher than the devices connected to the current Internet. Managing the data generated from these devices and their analysis is becoming more critical for application purposes.

vii. Security

Like any other technology, IoT devices, of course, they are vulnerable to security threats. As such, it will be a big mistake to not think about security issues as we gain several benefits from IoT. Internet of Things has a high level of transparency and privacy concerns. It is necessary to protect the endpoints, the networks and the data that is exchanged. This means there is a need to establish a framework for protection (security paradigm).

The above-highlighted characteristics have significantly contributed to the successful adoption and implementation of IoT technologies in existing information systems and applications, and have created value and support for human activities (Perera, *et al.*, 2014). This research study identified that IoT had been applied in different domains which include but not limited to smart cities, smart energy and electric grid, smart homes, intelligent buildings and infrastructure, smart health and has resulted in the creation of a “smart world” (Botta *et al.*, 2016) (Islam *et al.*, 2015) (Albishi *et al.*, 2017). How people live and work by saving time and organizational resources while bringing new opportunities for knowledge formation, innovation and development have entirely changed since the introduction of the “smart world” powered by IoT devices (Perera *et al.*, 2014) (Daj *et al.*, 2012) (Capossele *et al.*, 2016).

All things considered together with the human being’s desire to live in an automated smart world, novel IoT technology has gained, and it will continue to gain much attention in many diverse areas. In such an IoT ecosystem where these “things” are interconnected through a network, enormous and valid incomplete data is generated by IoT devices. The generated data needs to be processed and responded to in a short time. The leading cause of concern is the limited computational power, processing power and storage capabilities of IoT devices. Therefore, it is important to note that for successful adoption of IoT, it should be associated with a wide variety of other technologies.

2.3 Cloud Computing

Cloud computing was introduced and integrated into IoT to provide scalable and processing services to meet IoT demands since IoT “things” are largely resource constrained, exhibiting limited computational power, processing power and storage capabilities (Chen *et al.*, 2017). The term cloud computing was first used in 2006, precisely on the 9th of August by Eric Schmidt, Chairman and CEO of Google at the Search Engine Strategies Conference (Google Press, 2006). In 2011, the National Institute of Standards and Technology (NIST) defined cloud computing as “*a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction*” (The National Institute of Standards and Technology, 2011). The configurable computing resources in this definition include networks, servers, storage, applications, and services. The cloud model is made up of five vital characteristics, three service models and four deployment models, as summarized in **Figure 2-1** based on (The National Institute of Standards and Technology, 2011).

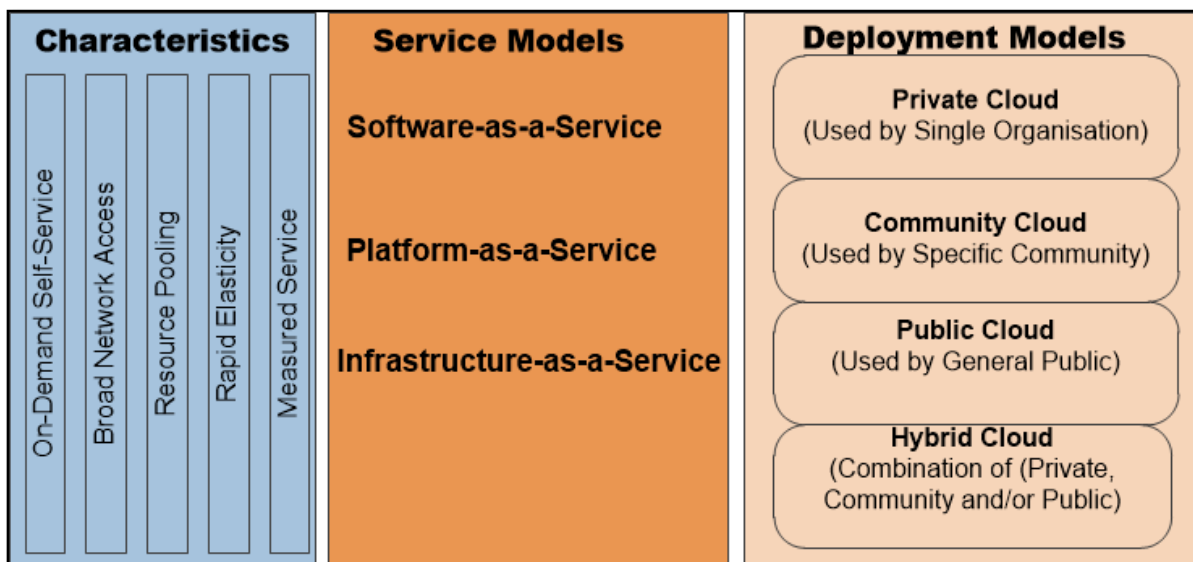


Figure 2-1: Cloud Computing Overview

Source: (The National Institute of Standards and Technology, 2011)

2.3.1 Cloud Computing Characteristics

As defined by NIST, there exist five vital characteristics which cover the most important aspects of cloud computing (The National Institute of Standards and Technology,

2011). The characteristics, as highlighted in **Figure 2-1** above, can be described as follows:

i. On-Demand Self-Service

Computing capabilities are offered to a customer automatically without any human interaction with the service provider as long as they are needed. The computing capabilities include but not limited to network storage, server time.

ii. Broad Network Access

Both thick and thin heterogeneous client platforms such as workstations, mobile phones, laptops and tablets which are connected to the internet can access cloud computing data centres through a standard mechanism. This is possible because cloud data centres are spread and available over the worlds network. Service providers such as Airbnb and Netflix already use and benefit from these centralised cloud resources.

iii. Resource Pooling

To avoid over-provisioning, or under-provisioning of resources such as memory, network bandwidth, processing and storage to the customers demand, cloud computing offers resource pooling. Resource pooling allows customers to utilise resources based on their demand dynamically. This will enable users to access and use cloud resources wherever they go.

iv. Rapid elasticity

This cloud computing characteristic allows horizontally and vertically scaling of resources with minimum management and configuration effort. Horizontal scaling allows the simultaneous serving of many clients, whereas the adaption of specific capabilities is handled by vertical scaling. This will enable customer demands to be dealt with efficiently while overall cost, energy consumption and resource-wasting are decreased.

v. Measured Service

Cloud providers always automatically measure, monitor and control the resources that will be used by customers such as bandwidth, processing and storage. This helps customers to know and track how the money they have paid via a pay-per-use basis was used. This is important as it provides transparency between customers and providers. Moreover, it will help in the measuring of QoS.

2.3.2 Service Models

The resources that are provisioned by cloud providers include a) *software services which are used via web browsers*, b) *developer platforms which are used to create and deploy cloud applications and finally*, c) *complete server infrastructure which handles virtual machines* (Micrac, 2008) *running on cloud resources*. As explained by Micrac, (2008) and supported by Baun *et al.*, (2011), the cloud resource delivery uses three service models, namely:

a) *Software-as-a-service (SaaS)*

SaaS is the first model, and as such, it is most restricted. Customers cannot configure or manage physical cloud resources. There are many SaaS service models, but the most popular used ones are Google Apps and Microsoft Office 365. All of the SaaS service models can be accessed via a programming interface or using a web browser.

b) *Platform-as-a-Service (PaaS)*

When compared to SaaS, PaaS is more flexible and comes with a development framework. PaaS offers a cloud environment that allows developers to develop, test and deploy applications. The most popularly used PaaS include Windows Azure and Google App Engine

c) *Infrastructure-as-a-Service (IaaS)*.

Infrastructure-as-a-Service is the most flexible model; as such, it allows the customer to deploy and run the virtual machines on the cloud physical resources. This gives the customer the power to control over operating systems, deployed application and storage. It should be noted that the customer will not have the ability to manage the underlying cloud infrastructure. The most popularly used IaaS is Open Stack and Amazon Web Services.

2.3.3 Deployment Models

Depending on their characteristics, the service models mentioned above are hosted in the following deployment models (Micrac, 2008) (Baun *et al.*, 2011).

- i. *Private Cloud*: Only a single organization which can be comprised of multiple consumers such as business units can have exclusive use for this kind of cloud infrastructure. The organization can own, manage and operate it. In some instances, a third party can also do the same. It could be on or off the premises.

- ii. *Community Cloud*: A community of customers with shared concerns such as policy, security requirements and compliance considerations can have exclusive use for this kind of cloud infrastructure. In this case, the community can be made up of different organizations, and they can operate and manage it. It could be on or off the premises.
- iii. *Public Cloud*: Contrary to the two above, which come with restrictions on who use them, the public cloud infrastructure is open to the general public. A government, business or academic organization can own, manage and operate it. It exists at cloud provider premises.
- iv. *Hybrid Cloud*: This cloud infrastructure is made up of private, community or public cloud infrastructures. Even though they are bound by proprietary technologies and standards that facilitate both data and application portability, they remain unique entities.

2.3.4 Virtualization

Another essential trait of cloud computing which makes it ideal to be used is its ability to use the concept of *virtualisation*. Virtualisation can be defined as “*the abstraction of the physical hardware resources of a computer system*” (Baun *et al.*, 2011). Virtualisation allows single physical hardware to be shared and used by multiple customers. Each customer will have their processing, storage and memory request treated differently in a separate way from the other user even though they will be using the same physical hardware. One of the significant advantages of virtualisation is it promotes the economical and efficient use of resources. It is important to note that there exist many virtualization concepts (Baun *et al.*, 2011); however, in this research, full virtualization and container virtualization, also known as Operating System (OS) virtualization are considered.

In full virtualisation, a virtual copy with specified CPU, RAM and other capabilities are packed in a virtual machine. The physical resources in one computer will determine the number of virtual machines to be created. Since each virtual machine is treated as a unique stand-alone machine, separate and different operating systems can be installed in each virtual machine on the same physical computer (Baun *et al.*, 2011). To isolate, monitor, manage, deploy and uniquely identify each virtual machine from the other, a virtual machine monitor middleware is used. The main advantage of full

virtualisation is that it allows elasticity of deployment, promotes secure provisioning of adaptable resources and offers the ability to compose heterogeneous hardware. One major challenge of full virtualization is that it requires a big amount of storage space and is also faced by long starting times which can seriously affect time-sensitive application (*Choice Reviews Online*, 2012). VMWare is such an example of an enterprise used for developing virtualization solutions.

Contrariwise to the full virtualization, container virtualization is regarded as very light-weight virtualisation and is built on top of the existing OS. The containers have separate and isolated virtual environment but run on top of the same host OS while using the same kernel and general physical hardware. This allows the container to be independent, thus allowing its access, use of storage space and processing on the general physical hardware. A Docker is one such example of a container technology .A Docker helps in distinguishing Docker Containers and Docker Images (Soltesz *et al.*, 2007). The Docker Container is made up of lightweight OS (base image), files added by the user and meta-data.

In contrast, Docker image is made up of several layers saving a snapshot of a Docker Container. The union file system in the Docker is used to merge several layers to make one image which is instantiated by the Docker runtime and results in a deployed Docker Container. When compared to VMs, containers take less time to start since they are using the same OS for every container. Moreover, containers can be deployed, released and updated very fast since they are lightweight (Soltesz *et al.*, 2007). Another advantage of containers is their ability to be combined with *Micro Services Architecture* (MSA) which are used in the designing of small, independent, light-weight and distributed software components. A microservice which is just a software component can be deployed quickly in a standalone container. Microservices brings a lot of advantages to their combination with containers as they are resilient, technology heterogeneity, easy to optimize for replacement, scaling and easy of deployment (Newman, 2015). Due to containers advantages as highlighted above, they can bring a lot of benefits when implemented in a fog computing framework as they are likely to support time-sensitive applications when compared to VMs.

As a result of the cloud computing characteristics mentioned above for service models, deployment models and ability to use the concept of virtualization, it has been widely

adopted in many businesses for backup, file storage, cost-cutting in terms of infrastructure, development and testing.

2.3.5 IoT-Cloud Architecture

Cloud computing characteristics, service models and deployment models made it possible for cloud computing to be adopted as the central computational backbone for IoT devices and form IoT-Cloud Architecture. **Figure 2-2** presents the IoT- Cloud architecture.

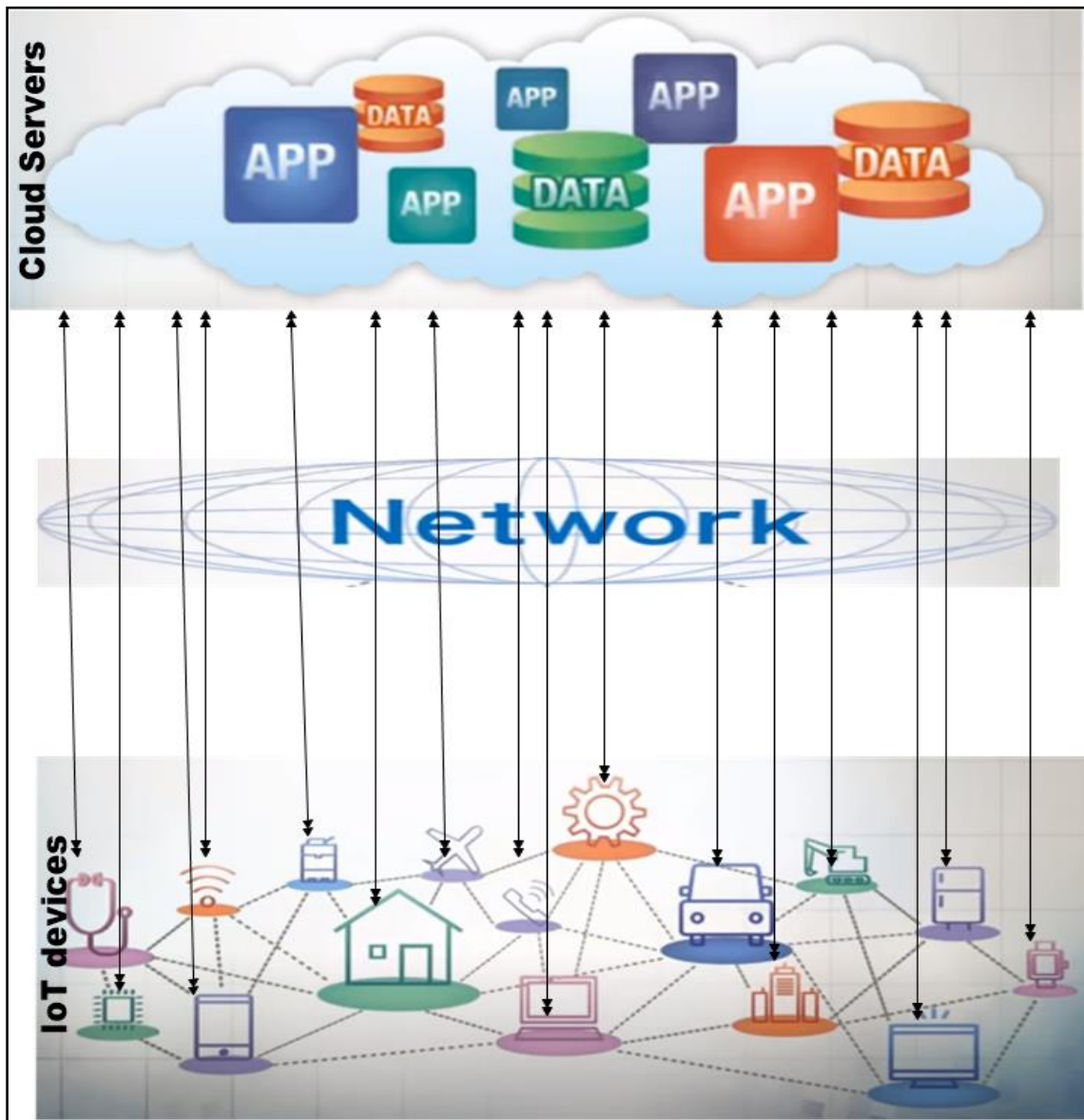


Figure 2-2: IoT-Cloud Architecture

Source: (Atlam et al., 2018)

In an IoT-Cloud architecture (**Figure 2-2**), the tasks of different types are generated by IoT devices are processed either in devices themselves or are sent to the cloud. All the tasks which require more computational power and storage space, which cannot be offered by resource-constrained IoT devices are sent to the cloud servers in the cloud layer. The cloud servers would process the tasks and send the output back to the IoT device. Thus, cloud computing provides scalable storage and processing services for IoT demands (Atlam *et al.*, 2018). In such a setup, IoT devices are geographically far away from cloud computing servers. Researchers argued that the integration of cloud and IoT requires more bandwidth, experience more latency and security can be compromised. Sending enormous tasks (data) created by IoT devices to and from the cloud requires exceptionally high network bandwidth (Atlam *et al.*, 2018). Explicitly, the unprecedented amount of data produced by IoT (sensors and other devices) burden the network resulting in network transmission delays (Dastjerdi and Buyya, 2016). These transmission delays will lead to high latency that can compromise QoS (Satria *et al.*, 2017). As a result, communication delays will be experienced due to unstable and intermittent network connectivity. Moreover, cloud computing does not have location awareness to the geo-distributed IoT devices, which also affect how IoT devices communicate with cloud servers. With the expected 50 billion deployments of the smart interconnected device such as mobile devices and intelligent sensors serving by the end of 2020, different vertical markets will be compromised in terms of bandwidth, latency and security of the data.

In a nutshell, an increase in transmission latency will increase round-trip time which affects response time and stress up the user who is depending on the IoT device output. On top of that, the processing speed in this environment mostly relies on the performance of users IoT device. This will affect IoT services that require low and predictable latency. As such, these challenges can only be addressed by an adaptive, geo-distributed and decentralized computational paradigms that complement the centralized cloud computing model by bringing cloud resource closer to the IoT devices.

2.4 Fog Computing

Owing to the cloud computing challenges mentioned above, several architectures which include geo-distributed cloud computing, physical proximity-based cloud computing, edge computing or fog computing, and mobile edge computing (MEC), were proposed (Chang *et al.*, 2017). Nevertheless, fog computing has gained much attention both in academia and industries. Fog computing was introduced by Cisco in 2012, not to supersede the cloud but to complement cloud by bringing the cloud resources closer to the “ground” (Bonomi *et al.*, 2012) (Vaquero and Rodero-Merino, 2014). OpenFog Consortium is an IEEE standard that defines fog computing as “a horizontal, system-level architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing continuum” (OpenFog Consortium Architecture Working Group, 2017). Fog computing architecture consists of a fog layer (physical or virtual) residing between smart end-devices and centralized (cloud) services, which bring both computational and storage capabilities closer to the ground where IoT devices reside (**Figure 2-3**). In other words, a fog layer bridge the geographical distance between IoT devices and the cloud layer. Fog layer nodes can help in filtering and pre-processing of data, processing of tasks and storage. Additionally, it provides for end-devices, local computing resources and, when needed, network connectivity to centralized services (Lorga *et al.*, 2018). As a result of fog layer nodes proximity to the IoT devices, latency is reduced, which helps in the minimization of the request-response time from-to supported applications. Moreover, data sent over the network will be reduced, which help in saving network bandwidth. It is important to note in the fog layer; communication can be done with Bluetooth, Ethernet, Wi-Fi, cellular network or any of the other communication mediums, which also helps in saving bandwidth. Fog layer nodes are also geographically distributed, which makes them location awareness. This is a vital trait that helps fog computing in offering reliable service execution and supporting even moving devices such as drones, smart cars, mobile phone without compromising QoS.

2.4.1 Fog Computing Characteristics

Fog computing can provide the above-highlighted advantages because of the following characteristics (Patel *et al.*, 2016) (Bonomi *et al.*, 2014):

- i. *Edge location, location-awareness and low latency.*

Since fog nodes are located at the edge of the network closer to the edge devices (IoT devices) which they intend to serve, it is easier to communicate with edge devices and be aware of their location. This helps in serving applications such as augmented reality, video streaming and real-time monitoring systems that are time-sensitive and do not tolerate high latency. Thus, fog offers edge location, location awareness and provide low latency between the user (endpoint devices) and fog nodes (Saad, 2018).

ii. Geographical distribution

In nature, fog computing nodes are geo-distributed when compared to central cloud computing. Their geo-distribution nature allows them to support and offer high-quality streaming to moving objects such as drones, vehicles and other mobile devices. This trait is fundamental in smart cities, smart industries.

iii. Mobility Support

Fog computing is well known for supporting both static and dynamic computation because of its ability to restructure the network topology. Geo-distributed fog computing nodes support the restructuring of the network topology. This is important in application areas like smart cities where cars, trains and other mobile devices should move from one point to another without their task executions being compromised.

iv. Large Scale Sensor Networks

Fog computing is characterized by large-scale sensor networks, especially in monitoring applications such as the smart grid. In such a monitoring environment, sensor networks always communicate with fog nodes requesting both computing and storage resources. This implies that in the fog computing environment, there should be a large number of geo-distributed fog nodes to support such sensor networks.

v. Save bandwidth

As a result of constant communication highlighted in (iv) above, between sensor networks and fog nodes, it means bandwidth is needed. In fog computing, there is a predominance of several wireless access such as Bluetooth and NFC, which helps to save the bandwidth.

vi. Real-Time Interactions

Unlike cloud computing, fog computing is meant to deal with tasks that are not CPU intensive as they have limited computational power. As such, fog nodes don't do batch processing but preferably real-time interactions which support time-sensitive tasks.

vii. Device Heterogeneity

Since the devices come from different manufactures, it means there is no standard interface, functionality and or deployment. Fog computing can cooperate these devices from various providers and enable communication when handling the request for processing from such devices.

2.4.2 IoT-Fog-Cloud Architecture

With the introduction of fog computing in the IoT-Cloud architecture, IoT-Fog-Cloud architecture was formed, as shown in **Figure 2-3**.

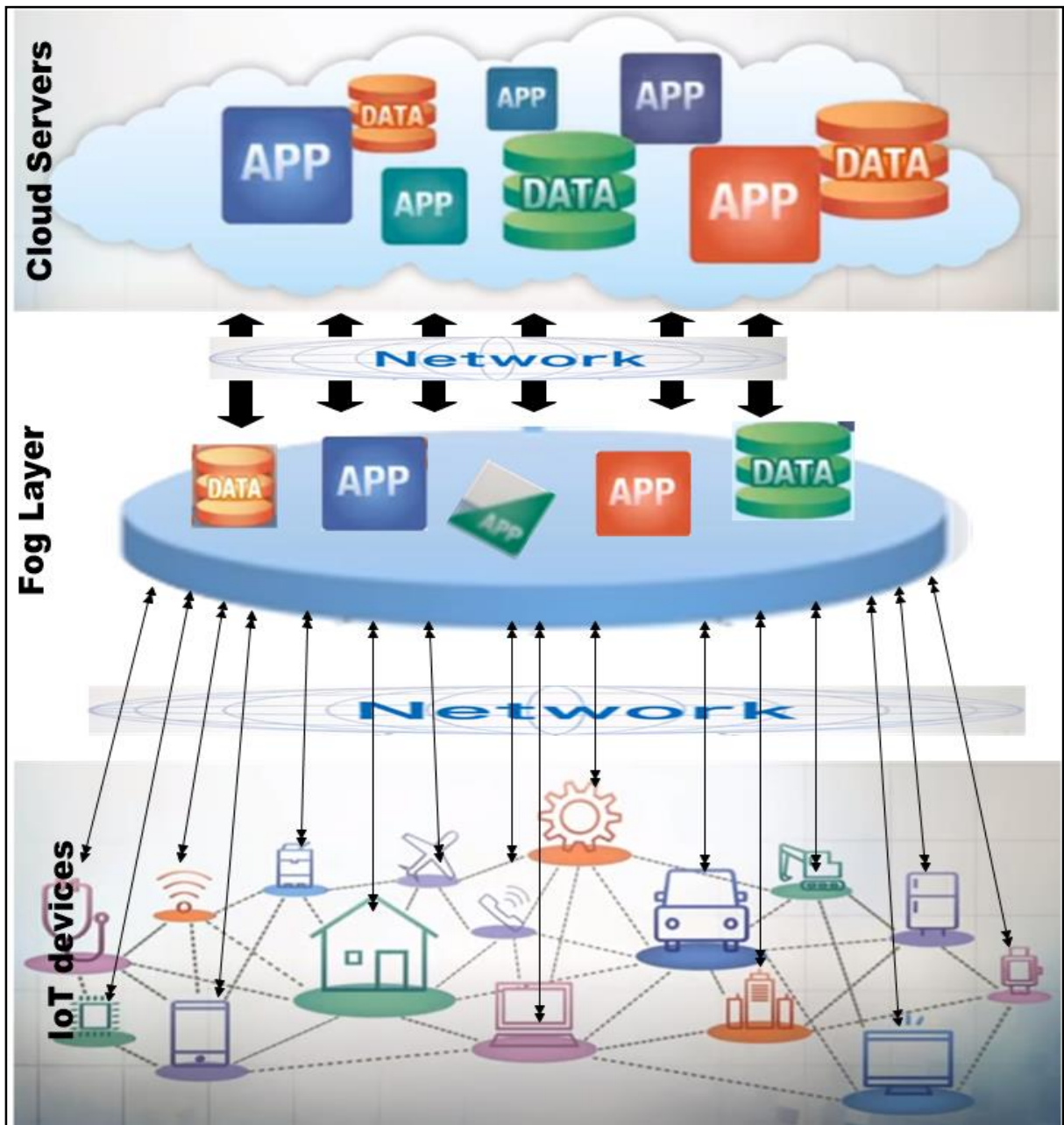


Figure 2-3: IoT-Fog-Cloud Architecture

Source: (Skarlat *et al.*, 2016)

In this IoT-Fog-Cloud architecture, the tasks of different types which are generated by IoT devices are processed either in devices themselves or offloaded to other computing devices in the fog layer nodes depending on the availability of resources in the fog layer. The fog node will process all the task, or part of the tasks and offload the other part of the task to the other fog node. If the tasks to be executed requires more computational power than the one available and offered in the fog layer, the reasoning component in the fog layer will offload the part of the tasks or the whole task to the cloud.

Due to this architecture (**Figure 2-3**), fog computing can support critical IoT services and applications to have improved QoS (Atlam *et al.*, 2018). Fog computing can be applied to, and support different application areas such as smart cities, smart homes and many other smart environments as highlighted by (Bonomi *et al.*, 2014).

As summarised by Chang *et al.*, (2016), fog computing is able to complement the cloud and help the IoT to exploit its potential because of its ability to offer SCALE (Security, Cognition, Agility, Latency, and Efficiency). Researchers Lorga *et al.*, (2018), Saad, (2018) and Luan *et al.*, (2015) shared the same sentiments and highlighted that fog computing supports low latency and location awareness, wide-spread geographical distribution, mobility support and device heterogeneity.

For the reason that fog computing offers the above-stated advantages, many researchers both in academia and industries have implemented fog computing in existing systems. The main goal is to benefit from fog computing characteristics, thereby improving existing system functions whilst improving QoS. It can be noted in the literature that fog computing has been successfully implemented in existing systems, and many application areas have benefitted significantly. The focus of many researchers now is to come up with strategies, methods and or procedures of improving fog computing itself. Such improvements will help in strengthening fog computing advantages in minimizing transit delay, improving availability, throughput and priority which are all QoS aspects.

2.5 Application Area : Smart Home

Because of fog computing characteristics and advantages, it has been implemented in many application areas with the motive to complement cloud computing and offer low latency and improved QoS. Recently, connected vehicles, smart cities, smart grid and smart home are some of the application areas where fog computing has been used (Yi *et al.*, 2016)(Naha *et al.*, 2018). For this research, the application and testing area will be smart homes.

The need for automation and security in homes has necessitated the adoption and creation of “smart homes” across the globe. Smartphones are the major driving force in the implementation of smart homes as they are used to control most of the household gadgets (Nachiket, 2019). Many house appliances can be connected, including smart laundry appliances, smart water heaters, water treatment appliances, kitchen appliances, intelligent compactors, smart compactors, smart air purifiers and filters. Based on technology, connected home appliances can be connected using Wi-Fi, NFC, Bluetooth technologies. Of the named technologies, Wi-Fi and Bluetooth are the most used technologies in the connected home appliances.

The work of Biljana and Kire, (2016) pointed out some of the challenges that are faced in implementing the IoT in a smart home as far as avoiding latency issues is concerned. Some of the obstacles hampering the growth of connected home appliances, especially in Africa as compared to other continents, include low penetration of the internet infrastructure and the affordability issues of bandwidth. Fog computing seems like a solution for addressing some of the problems in the adoption of smart homes. It is a fact that QoS is vital in a smart home setup because most of the gadgets require real-time response. Having enhanced QoS in smart homes will make connected homes safe and secure to leave in.

2.6 Quality of Service

It is essential to realize that Quality of Service is a crucial service requirement. Providing satisfactory QoS is a fundamental goal in general networking, fog computing, cloud services or in general information systems. Depending on the perspective, QoS can have several definitions.

From a networking perspective, QoS is defined as “*the network capability to deliver enhanced service to designated network traffic over numerous technologies such as Asynchronous Transfer Mode (ATM), Frame Relay, SONET, Ethernet, and 802.1 networks, and IP-routed networks that may use any or all of these underlying technologies*” (Cisco, 2014).

In general information systems, QoS can be defined as “*the ability to provide different priority to different applications, users, or data flows or to guarantee a certain level of performance to a data flow*”.

In cloud and fog computing, QoS is “*non-functional properties of cloud /fog services, which describe how well a service is performed, such as compliance, availability, reliability, responsiveness, price, security, latency*” (Zheng *et al.*, 2017).

In the light of the definitions above, QoS is a significant factor because it ensures improved services for the end-user. In the case of fog computing, QoS is a very crucial service requirement needed to promote reliability, improve throughput, reduces energy consumption, minimize network delays and latency (Naha *et al.*, 2018).

Correspondingly, studies have been done both in academia and industrial domain on how fog computing technologies can be used and implemented in existing systems to enhance QoS. It was from the literature review that we identified the different methods and strategies which were used in fog computing framework to minimize transit delay (TD), improve availability (AV), throughput (TP) and giving priority (PO) which are all QoS aspects.

Failure to maintain or improve QoS in fog computing seriously affects fog computing-based systems /applications. This will cause fog computing-based systems/ applications to encounter end to end communication delays (Souza *et al.*, 2017), service migration issues (Song *et al.*, 2017), workload deployment challenges (Taneja and Davy, 2017), computation and resource allocation problems (Wang *et al.*, 2017). For this reason, it is vital to always maintain high QoS in fog computing systems, especially with the incoming of IoT devices whose tasks are mostly time-sensitive.

2.7 Related Work

This section looks at resource provisioning/allocation and scheduling in a fog computing environment.

2.7.1 Resource Provisioning and Scheduling for Fog Computing Environment.

Resource provisioning is defined as “*the procedure to orchestrate, allocate, deallocate, reallocate tasks to resources and monitor all available system resources*” (Kalyvianaki, 2008). It is vital to utilise all fog layer resources effectively and efficiently since the fog layer resources have less processing power when compared to the cloud (Li *et al.*, 2017). It is crucial to note that when doing resource provisioning, the main goal should be to reduce round-trip time (latency) and also improve QoS. If resource provisioning is done well, transit delays will be minimized, throughput and performance of a system will be enhanced, which leads to improved QoS. Resource provisioning can be either reactive or proactive. Proactive resource provisioning allows the system to act earlier before an event happens using a predictive strategy. Whereas, a reactive resource provisioning waits for something to happen then react to the situation.

Many resource provisioning researches has been done and implemented successfully in cloud computing and mobile cloud computing (MCC) as indicated in the surveys done by Zhan *et al.*, (2015), Singh and Chana, (2015) and Dinh *et al.*, (2013), Lu *et al.*, (2015) respectively. However, the strategies described in these works cannot be adopted directly in fog computing due to the dynamic and heterogeneous nature of fog landscape. Moreover, the cloud computing and MCC resource provisioning strategies suggested does not put into consideration ways to reduce latency, task execution times and allow close-range communication. This affects as it increases the bandwidth cost and also the cost of using the cloud. Nonetheless, it was from cloud computing and MCC resource provisioning researches where ideas for resource provisioning were borrowed, improved and implemented in fog computing. Literature shows that several types of research have been done as far as resource provisioning and scheduling are concerned. In fog computing literature, resource provision can also be placement or resource allocation and is made to minimize latency and improve QoS (Keller *et al.*, 2012).

The work of Ni *et al.*,(2018) proposed a resource allocation strategy for fog computing using Priced Timed Petri Nets (PTPN), which helped to utilize and link both cloud and

fog resources. Their approach helped to improve the efficiency of resource utilization, satisfy user QoS requirements and maximize the profit of both providers and users, which has become a big challenge. Priced Timed Petri Nets technologies allowed the user to choose the satisfying resources autonomously from a group of pre-allocated resources. Based on the results, the authors concluded that their approach could be more efficient when compared to static allocation strategies based on task completion time and price.

Researchers in Yang *et al.*, (2018) introduced a novel dynamic resource allocation framework to incur minimum operational charge while satisfying the applications' latency requirements. Their results ensure that the service response was minimized while achieving up to 33% operational cost reduction when compared to the fixed-location practices. Their strategy did not consider the cost in situations where there is a need to migrate from user-to MEC assignments.

Aazam *et al.* proposed a dynamic resource provisioning strategy which used fog micro datacentres in Aazam and Huh, (2015a) and Aazam and Huh, (2015b). Their setup was almost the same as that of Skarlat *et al.*, (2016) which had IoT devices, fog landscape and the cloud. The only difference was that in each layer, there were different micro datacentres which helped to orchestrate fog cells. On their prediction resource management model, which was theoretical, they considered resource demands based on the type of accessing device and relinquishing probabilities which are informed by historical pricing models, access data and service types. One of the weaknesses of this strategy was that it could not react in the dynamic fog landscape changes, which calls for improvement (Aazam *et al.*, 2016).

The work of G. Li *et al.*, (2018) put forward a method of resource estimation that would help in choosing available resources, select the suitable resource to meet the needs of users. Their approach classified and matched resources according to the weighted Euclidean distance similarity. To correct similarity matching function, two strategies, namely penalty factor and Grey incidence matrix, were used. A Regression-Markov chain prediction method was used to analyze the change of the load state of the candidate resources and select a suitable resource. The simulation was used to validate the effectiveness of the estimation method. Precision and recall were used as benchmarks to test for the performance. Their approach helped in addressing

decreased QoS in fog computing due to increased network devices and cloud center load, which was causing a delay, thus affecting timely response. Even though they manage to come up with a resource allocation method, they did not consider resource estimation to balance the satisfaction between the two key player's which are users and service providers. Two factors that were pinpointed to defend their argument was that the QoS attributes system are extensible, and the user QoS requirements are dynamic.

With the same motive of addressing the challenge fog service placement, Skarlat *et al.* (2016)(a) proposed a conceptual framework for fog resource leasing and releasing (provisioning) (Skarlat *et al.*, 2016). The envisioned architecture was evaluated using a customized simulation. It was observed that the approach decreased task request delays by 39%. In their research, they did not implement it on a real-world testbed which made it difficult to do a systematic evaluation when considering real-world network data such as bandwidth and delays. In a bid to address some shortfalls highlighted in their work above, in 2017, Skarlat *et al.* (b) implemented the system in iFogSim testbed as to solve the Fog Service Placement Problem (FSPP) while considering the heterogeneity of applications and resources in terms of QoS attributes (Skarlat *et al.*, 2017). They introduced a generic algorithm which assisted in reducing network communication delays and promoted a better utilization of fog resources. Their work created a theoretical and practical foundation for fog resource provisioning and service placement, then named the framework FSPP. Simulation results showed an improvement in service placement plan produced by the genetic algorithm, greedy first-fit heuristic, and an exact optimization method. For future work, they recommended testing the FSPP solution in a real-world setup. In a different work titled, "Towards QoS-aware Fog Service Placement", the same author Skarlat *et al.* (c) tested the FSPP in a real-world setup. It helped in determining an optional mapping between IoT applications and computational resources to optimize the fog landscape utilization while satisfying the QoS requirement of the application (Skarlat *et al.*, 2017). Accordingly, if a task is submitted to a control node, the control node will check whether it requires a cloud or fog node and that decision will also be made based on the execution time required. After that decision is made, if it requires a fog colony, it will be given to the nearest fog colony based on the computational resource available. If the assigned fog colony does not have enough resources, then the control node would

send the task to the accessible neighbour colony or the cloud. The approach prioritized those with higher priority to be given first preference. Their experimental results showed that the FSPP utilizes the fog landscape for 70% of services, thus reducing the execution cost. It should be emphasized that the approach was tested in a real-world test-bed which manages to obtain realistic data such as communication link delay for service placement evaluations in a fog landscape compared to most approaches. However, their approach when considering the colony to give a task, they looked for the closest neighbor colony but did not put into consideration to find the most efficient neighbor colony which might affect QoS.

In their quest to provide efficient utilization of network resources and minimize application latency in the IoT ecosystem, Taneja and Davy, (2017) introduced a resource-aware placement for IoT application modules in Fog-Cloud Paradigm. Their work was motivated by the fact that fog nodes are not computationally powerful enough to host all the modules of an application in the IoT ecosystem context. They introduced a module mapping algorithm in a three-tier setup which has different computational capacity. Each tier would be assigned to support a specific component of the application based on its level of computational capacity. Those who would require higher computational power were assigned to the fog node with the necessary resource capacity, thus promoting efficient resource utilization which in turn brings quick processing of the application, therefore reduces the completion time of processing a request. When compared to other traditional cloud placement approach, their approach proved beyond certain doubt that in the future IoT application and even future needs, it will address latency-sensitive needs. The authors also reiterated that their work made some positive strides towards resource scheduling in fog devices; more work still needs to be done as to assist in coming up with scheduling policies. Moreover, though the algorithms managed to improve on network usage effectively, energy consumption and response time, the researchers put a blind eye on issues of network connectivity, failure of nodes. These two play a critical role in attaining better QoS.

To assign tasks to servers according to the latest network and server status in an efficient way, Rashidi and Sharifian, (2017) proposed ANFIS (Adaptive Neuro-Fuzzy Inference System) algorithm. The approach helped in distributing user request to

different cloudlets, which resulted in achieving higher utilization of resources and efficiently deal with network dynamics which in return improved user quality of service.

Researchers Aral and Brandic, (2017) were also of the view that effective resource allocation and network utilization alone will not completely address the QoS issue in fog computing unless or otherwise we look and address other challenges faced in edge data centers. Edge data centers mostly suffer from hardware and or software failures due to lack of advanced support systems such as power generators and air conditioners which affect them to work effectively and efficiently. As such, these failures have a negative impact in terms of response latency and bandwidth traffic since failed edge data centers affect the distribution of tasks to other fog nodes or cloud data centers. This will compromise everything in the network or other resources leading to a delay in response time which is critical for edge applications. As such, the researchers reiterated the need to have proactive algorithms/ models which accurately predict the availability of a virtual machine before assigning a task to that resource. The work of Aral and Brandic, (2017) proposed a Bayesian Network Model of QoS to address the above-mentioned predicament. They aimed to estimate the availability level of a VM then channel all QoS related parameters to the one available. They compared their model with other machine learning methods such as Naïve Bayes and Logistic regression. They discovered that their proposed method obtained 94% accuracy and 44% of decreased SLO violation compared to the other two.

The work of Li *et al.*, (2017) is one such which echoed the same sentiments that fog nodes have limited resources when it comes to processing power. Hence, they can quickly become overloaded when a large amount of user's request arrives during peak hours resulting in processing delays which in-turn will affect QoS. As such, they proposed two resource management schemes that are Fog Reservation (FR) and Fog Resource Reallocation (FRR). These approaches would reserve some fog nodes so that they will be used only by real-time services. Moreover, if there was overload in those reserved nodes, fog reserved for low priority services were reallocated to be used by high priority vehicular services. Experimental results showed an improved one hop access probability for real-time vehicular service. Even if the fog resources were under heavy load, it managed to achieve low delay. However, also the approach brought out results which were favourable in vehicular context, low priority jobs were

sacrificed and as such their QoS was compromised which can be a disadvantage because they have to wait much longer for the response.

The work of Xiao and Krunz, (2017), proposed a novel offload forwarding strategy. Where fog nodes would either not offload or offload and forward part or its entire load to be processed by other local fog nodes which are idle and have better computational power than them. This strategy helped to minimize the average response time which included workload transmission time and queuing delay at the fog layer and significantly improved the performance of fog computing network, thus improving Quality of Experience (QoE) of users. The researchers validated their approach using traditional ADMM approach, which proved that it could not be used to solve the offload allocation problem for fog computing. However, their work used fewer nodes and ignored time-critical events which call for further experimental trials to check how it will perform in such a scenario.

In Song *et al.*, (2017), they applied a QoS-based approach task distribution in edge computing networks. Their approach was almost similar to the works of Xiao and Krunz, (2017), with the only difference being that their approach was applied in IoT applications. Furthermore, it would periodically distribute incoming task in the edge computing network. They reasoned that there would be an increase in processed tasks in an edge computing network. As supported by their results, the QoS requirements of the task completed in the network were satisfied. In their experiments, they applied two approaches with a local approach which would execute its task locally on its access node. In contrast, the random method would select an eligible node for each task as its execution node where all the QoS requirements are satisfied. Song *et al.*, (2017) approach did not give high priority to time-critical tasks with different operation software to test for Interoperability. The result proved to be sound as they used more nodes than (Xiao and Krunz, 2017).

Inspired by the goal of minimizing resource consumption which reduces energy consumption and carbon emission rate through load balancing, Neto *et al.*, (2017) introduced a Multi-tenant Load Distribution Algorithm in Fog environments (MtLDF). This algorithm was implemented in a fog environment for sharing the load among nodes where they considered delay and priority as special individual blocks of multi-tenancy requirement. Their approach adopted two Tenant Maximum-Able Delay

(TMAD), which would address QoS and Tenant Priority (TP) which was responsible for selecting and grouping tenants in terms of their computational power importance. The approach had a Fog Management Layer (FML) which would send load to an eligible fog node considering load balancing. The experimental results proved that MtLDF could improve load distribution better compared to Delay-Driven Load Distribution strategy that was once used. Resource utilization also improved to a maximum of 97%. However, this approach needs to be improved if it is to be used in IoT time-critical devices which require at least 99% distribution value if we are to achieve higher QoS

The work of Wang *et al.*, (2017) highlighted that resource sharing and prioritization plays a pivotal role to obtain high QoS in fog computing. Their results and evaluations pinpointed that prioritizing execution can minimize delay for tasks with higher priority. Latency can be reduced by sharing operations of lower priority processing, and throughput of valid temporary events can be decreased through shedding. These conclusions were drawn when they implemented a system in cyber systems which require high QoS such as bounded latency which is a critical factor in cyber-physical applications. Wang *et al.*, (2017) developed middleware for real-time Cyber-Physical Event processing (CPEP). The CPEP was able to configure processing operations, processing prioritization and sharing and enforcing of temporal validity and shedding. They evaluated its effectiveness in terms of prioritization, sharing and shedding; and validated it using Data Distribution Service, which is also another messaging middleware.

Although offloading and resource allocation has made better strides in addressing QoS in Edge computing in other application areas as supported by previous researchers in above literature, contrary, Zhang and Zhu, (2017) argued that mobile data offloading in ECN might impose new QoS guarantee glitches. Their argument was based on the upcoming of 5G networks. They pinpointed out the new challenges are encountered when users request multimedia services which are sensitive to time and demand a lot of bandwidth which might become difficult in supporting the statistical delay-bounded QoS. To address the highlighted problems, they proposed a statistical delay bounded QoS provisioning schemes. The statistical delay bounded QoS provisioning schemes used the effective capacity theory for two types of mobile

offloading that is Wi-Fi and Device-to-Device offloading: where they formulate the Wi-Fi and D2D offloading as one hop and two hops respectively. When they validated their approach through numerical analysis, their system proved that it could be used as a benchmark to address the problem mentioned above.

Based on the piloted literature review, it is clear that many strategies have been applied in fog computing to address issues of tasks to resource allocation while minimizing latency and improving QoS. Bearing in mind of the 50 billion expected devices to be connected by the end of 2020, it is vital to keep finding more strategies on improving fog computing as far as resource allocation is concerned (Ericsson, 2011).

2.7.2 Scheduling techniques

Task scheduling plays a critical role as far as reducing latency and improving QoS in fog computing is concerned. This was supported by a systematic literature review done by Yang and Rahmani, (2020), on checking task scheduling mechanisms in fog computing. Task scheduling in fog computing can be defined as the effective and efficient assignment of IoT tasks to fog layer resources (Mtshali *et al.*, 2019). Effective and efficient task scheduling means not to over-consume the limited available resources at the fog layer. If fog nodes are over consumed, this can lead to application failure or network breakdown, leading to an increase in latency which has a negative impact on real-time applications (Mtshali *et al.*, 2019). Task scheduling can be either static or dynamic. In static scheduling, the first step is to know the complete system information and resource mapping before tasks are executed (Choudhari *et al.*, 2018). Contrarywise, in dynamic scheduling, resource assignment depends on the system current state, computer machines and tasks submitted before scheduling decisions are made.

From the reviewed literature, most of the existing solutions used several different approaches to do resource provisioning and scheduling procedures. It can be deduced from the literature that dynamic programming, in particular, linear programming is the most common one being used for the formulation of the resource provisioning and scheduling problem. The majority of methods that are used to solve linear programming are either utilizing heuristic algorithm or exact mathematical methods (Rothlauf, 2011) (Chaisiri *et al.*, 2012) (Beheshti and Shamsuddin, 2013).

The reason why heuristics algorithms are mostly used is that they can resolve problematic issues in a faster way when compared to meta-heuristic algorithms which are considered as too slow in terms of performance. Moreover, heuristic algorithms are known to provide optimum solutions, while in most cases, meta-heuristic algorithms usually fail to discover optimal or precise solutions (Müller-Merbach, 1981) (Müller-Merbach, 1985).

According to Syed *et al.*, the six most commonly used heuristic algorithms for scheduling in cloud computing and which are adopted in most fog computing solutions as proved from literature above are: “Minimum Completion Time (MCT), Minimum Execution Time (MET), Min-min, Max-min, Sufferage and First Come First Serve” (Madni *et al.*, 2017).

- **Minimum Completion Time (MCT):** Tasks are assigned to VMs or resources based on task completion predictable time in random order. Meaning tasks are assigned to a VM or resource with the earliest time of completion. In some cases, MCT allocates tasks to VMs with no minimum execution time.
- **Minimum Execution Time (MET):** Tasks are assigned to VMs or resources based on tasks best predictable time without considering whether a resource is available or not. In this case, the algorithm only considers the minimum execution time of the tasks. In some cases, this results in load imbalances since the VMs or resources to be assigned would not have been considered whether they can handle the task or not (Braun *et al.*, 2001).
- **Min-min:** The algorithm checks for the minimum completion time of all the tasks concerning the available machines, then a task with minimum completion time is removed from the group of un-scheduled tasks and is assigned to the subsequent machine. The process is repeated up until all the un-scheduled tasks are assigned to a machine (Aissi *et al.*, 2005). If tasks with minimum completion are many, then those with maximum completion will be faced by starvation problem (Braun *et al.*, 2001).
- **Max-min:** The algorithm checks for the maximum completion time of all the tasks in relation to the available machines, then a task with maximum completion time is removed from the group of un-scheduled tasks and is assigned to the subsequent machine. The process is repeated up until all the un-scheduled tasks with maximum completion are assigned to a machine

followed by those with minimum completion time (Aissi *et al.*, 2005). This approach is suitable when a few tasks have maximum completion time; otherwise, those tasks with minimum completion time will end up facing starvation problem (Braun *et al.*, 2001).

- **Sufferage:** In this case, tasks values with minimum and second minimum completion times are calculated first by the algorithm. Then, in the second stage, the difference in the task's values are considered. Those tasks with sufferage (minimum difference) are then allocated to the consistent VM. After that, the assigned task is removed from the un-assigned tasks group, and an update of resource availability is done. This is repeated up to the time when all tasks are assigned to a resource (Maheswaran *et al.*, 1999).
- **First Come First Serve (FCFS):** Processes that are scheduled and managed by FCFS algorithms are those that automatically execute tasks or resources in the order they arrive. The first task to arrive is the first task to be executed, thus following the first in first out (FIFO) concept. One of the advantages of this approach is it serves VMs or resources and time as it is regarded as a simple process of scheduling which is error-free and efficient (Madni *et al.*, 2017). As a result of its advantages, the FCFS algorithm is used by most simulators, such as iFog (Gupta *et al.*, 2017), CloudSim (Calheiros *et al.*, 2011) and GridSim (Higashino *et al.*, 2016).

In light of the above evidence, these algorithms perform differently in different application areas and depending on what needs to be done. For this study, the FCFS scheduling technique was adopted, modified and implemented in RAS to fulfil our research goal of doing research allocation and scheduling while minimizing round-trip time (latency).

2.7.3 Round-trip Time (Latency)

In general, latency can be defined as “*a measurement used for measuring delay*”. In the context of fog computing, latency can be regarded as “round-trip delay/time” and is considered as one of the most impact factor (Shukla *et al.*, 2019). Round-trip time can be defined as “*a measure of the time taken by a task or data to move to its destination and back to its original position over the network*” (Shukla *et al.*, 2019). Reducing round-trip time is vital in time-critical applications like health care, smart

homes and smart cities. When there is high round-trip time, the IoT requests are not processed and returned quickly, which seriously affect time-critical applications (Aazam and Huh, 2015b). Even if the tasks response is returned late, it will be rendered meaningless, inadequate, unreliable by the real-time applications and end-users. Therefore, it is crucial to reduce round-trip time always. Round-trip time can be minimized by minimizing communication latency, computation latency, and network latency should be reduced (Shukla *et al.*, 2019). Our proposed solution seeks to reduce round-trip time by reducing queuing time and offloading time.

2.7.4 Software Frameworks

Several approaches have been used in literature when developing fog computing frameworks but lacked an extensible programming model as far as resource provisioning is concerned. To be able to design an adaptive resource allocation scheduler in fog computing framework for the IoT ecosystem, software framework approach is the most suitable for this research.

A software framework is regarded as a base structure for applications and service execution in a specific software environment. The reason for choosing the software framework approach is that it allows the developer to inherit some concepts from existing frameworks and modify them utilizing their application (Riehle, 2000). This is possible because of software framework have reusability characteristics. The reusability characteristic enables the developer to concentrate more on addressing the problem in this case resource provisioning aspect in fog frameworks, instead of implementing the environment basics.

According to Riehle, (2000), software frameworks can be either white-box and or black-box frameworks. White-box structures are supposed to be configured by extending explicit interfaces so that they can be executed in the chosen environment. Contrary, black-box structures are regarded as ready-to-use, and they are no need to continue or further development to be executed. Most frameworks are a combination of both the white-box and black-box structure. This thesis took a combination of the two in an adaptive resource allocation scheduler in fog computing framework for the IoT ecosystem.

This work borrows the ideas and concepts from the works of Kim and Lee, (2014) and Vögler *et al.*, (2016) who used software frameworks for IoT environments. The work of Kim and Lee, (2014) acts as a guiding framework that was used to develop, provide and execute applications using web-GUIs. In this work, specific aspects were added to cater for distributed application provisioning in the IoT landscapes, which were lacking in the framework of Kim and Lee, (2014). The work of Vögler *et al.*, (2016) introduced a scalable large-scale IoT framework that was implemented in a smart city environment. Some aspects were borrowed from the LEONORE framework developed by Vögler *et al.*, (2016). The LEONORE framework provided the knowledge on how infrastructure and toolset of IoT applications can be deployed at the edge of the network. The LEONORE framework set up some essential parameters to save bandwidth between the edge and the cloud, and enable a distributed and scalable IoT service deployment. This is the reason this research borrowed these concepts.

2.8 Research Gap in the Current Fog Computing Research Provisioning Strategies.

Even though positive strides have been made in addressing resource provisioning challenge while minimizing latency and improving QoS in fog computing, the suggested strategies still need improvement. It is important to note that all the proposed and implemented resource provisioning and scheduling strategies in literature, first consider whether there are available resources in the fog layer before sending a task. If there are available fog resources, they then assign and send the task to the fog layer. It is in the fog layer where the decisions are then made to either process the whole tasks, part of the task or offload the tasks to the next fog node or to the cloud which are idle and have better computational power than them. One drawback of this kind of resource provisioning approach is that, it sometimes wastes the resource-constrained fog layer resources and time in making decisions, especially if the fog layer is not going to process the task.

Secondly, sending both time-sensitive and non-sensitive tasks to fog layer without considering tasks QoS requirements, deadline requirements, and user needs; adds more unnecessary load to the fog layer. Especially, if the bulk of the tasks send are not time-sensitive tasks but are non-time-sensitive tasks that require more computational power which is only available at the cloud layer. This will affect time-

sensitive tasks as they will be competing for fog layer resources with non-time-sensitive tasks. This will result in time-sensitive tasks missing deadlines, and their output will end up not being of any use to the IoT device and end-users.

When doing resource provisioning, it is crucial to classify tasks as time-sensitive and non-time-sensitive tasks before assigning them to resources. This is because tasks have different computational and QoS requirements (Yang *et al.*, 2019) (Zhang *et al.*, 2019). Time-sensitive tasks should be given higher priority to either network resources or fog resources when compared to non-time-sensitive tasks. Prioritizing task will help in meeting deadlines for time-sensitive tasks. Time-sensitive tasks should always meet their deadline because failure to meet a deadline has a detrimental impact in critical applications like medical health applications, smart homes and smart cities where responses are needed in real-time.

Thus, with the current resource provisioning methods used, most time-sensitive tasks will end up missing their deadlines. This challenge is escalated with more IoT devices that are being connected daily to the network, which are adding additional burden to the network and resources. Resulting in high latency problems and resource challenges at the fog layer. Consequently, there is a demand to come up with a novel resource provisioning strategy and scheduling strategy in the IoT-Fog-Cloud architecture, whose responsibility is to flag and categorise tasks from IoT devices as either time-sensitive or non-time-sensitive tasks. Then, if the task is time-sensitive, it is assigned to the fog layer. If the tasks are not time-sensitive, it is sent directly to the cloud layer. This will help in saving fog layer resources and time which was not the case with the existing strategies as they use fog resources and time when deciding whether to process the whole task, part of the task or to offload it to the cloud layer completely. Moreover, the approach should also not starve those tasks that are not time-sensitive, as was the case with the strategy proposed by (Alnoman and Anpalagan, 2018).

Therefore, in the context of our proposed Resource Allocation Scheduler, when a task is received, it should be first classified as time-sensitive or non-time-sensitive. Then using the modified FCFS heuristic algorithm, classified tasks are assigned to either the fog layer for time-sensitive task or to the cloud layer for those tasks that are not time-sensitive, depending on the task request requirements. Resource Allocation

Scheduler is responsible for resource allocation and scheduling, giving high priority of fog resources to time-sensitive tasks. The RAS considers task deadlines, resource constraints. This research is of paramount importance as several application areas such as smart health, smart city, smart grids who require real-time response would benefit from the findings of this research.

2.9 Conclusion

This chapter gave a comprehensive insight into the IoT technology, cloud computing and fog computing which are the enabling technologies and foundation for this research. A review of several strategies and methods that have been suggested and implemented in fog computing to address resource allocation and scheduling challenges are presented. Firstly, the analysis of technologies and researches done in fog computing framework helped in giving an overall insight on functional, technical and non-functional requirements of fog computing framework in an IoT-Fog-Cloud architecture. Secondly, after a critical analysis of the literature, open research challenges which still exists in fog computing frameworks as far as reducing round-trip time and improving QoS is concerned, were highlighted. Open research gaps helped in justifying the importance of this research. This chapter helped in answering the first research question, “What are the key challenges in communication and computer resources allocation in an IoT environment?”. Moreover, helped in answering the second research question, “How are communication and computing resources allocated and assigned among the IoT devices based on tasks, requirements and priorities?”.

Chapter Three: Research Design and Methodology

3.1 Introduction

This chapter answers this study's third research question: “which approaches can be used to build a resource allocation scheduler framework for IoT environment?” The chapter, therefore, presents the procedures and techniques that were used for this study. The chapter is structured as follows: **Section 3.2** highlights the research methodology that was followed. **Sections 3.2.1, 3.2.2, 3.2.3 and 3.2.4** explains the four phases of the top-down methodology in the context of this research and what was done at each phase. The chapter conclusion is presented in **Section 3.3**.

3.2 Research Process Design

As stated in chapter one, this study aimed to design and implement an adaptive resource allocation scheduler in the fog computing framework for the IoT environment. The aim directly put this work within the communication networks domain, in the transport layer of the Open Systems Interconnection (OSI) model in particular. The communication networks domain has several standard methodologies for implementing framework designs. Although several conventional methods are used for implementing network design solutions, the most common approaches are the top-down approach and the bottom-up approach (ORACLE *et al.*, 2014). The top-down consists of the four phases: requirement analysis, logical network design, physical network design and testing, optimizing and documentation design. The bottom-up has similar phases with the top down, the only difference being that the latter's phases are the reverse order of the top-down approach. Although the bottom-up approach is generally faster as it is usually based on past projects, the methodology is known of having a higher probability of failure. It is also known of not taking into consideration all necessary service applications, and that may lead to design omissions that would necessitate a redesign of the framework. The top-down approach is very much ideal for designing and testing network solutions. It starts with a clear understanding of the goals of the required solution. It also allows planning for scalability and adaptability, which was among the desired milestones of the development of RAS. The top-down

methodology is also simple in terms of understanding and troubleshooting; flexible and allows the addition of new protocols and technologies at a rapid rate.

Furthermore, the top-down approach is a streamlined and systematic methodology that accommodates the increasing requirements of remote access, bandwidth, scalability, security and reliability in the context of IoT-Fog-Cloud architecture. Top-down network design is a standard methodology recommended by Cisco (Oracle *et al.*, 2014) and has been successfully used by many researchers in literature which include Mulyawan, (2011), Rasmila and Laksana, (2019) and Giovanni and Surantha, (2018). The works of Giovanni and Surantha, (2018) who used top-down network design resonates well with our study. Based on the reasoning above, this study, therefore, adopted the top-down methodology. **Table 3-1** shows the phases of the top-down approach as applied in this study.

Table 3-1: Top-Down Research Methodology Phases

Stage	Objective and Process	Milestone
Requirement Analysis	Problem Identification and Motivation *Systematic Scrutiny of Literature on Fog Computing Frameworks as to: -Identify Open Research Gaps in Fog Computing Frameworks. -Justify why it is Important to Address the Open Gaps. -Choose One Open Research Problem to Address. -Define Aim, Objectives and Research Questions. -Define Functional, Non-functional Requirements, Technical Specifications -Investigate the Tools to Use.	-Identified Open Research Gaps in Fog Computing Frameworks. -Defined Why it Is Important to Address the Open Gaps -Resource Provisioning as the Core-Problem to be Addressed. -Resource Provisioning to Minimize Round-trip Time and Improve QoS. -Literature Review Paper Submitted in International Journal for Fog Computing.
Logical Network Design	Designing a Network Topology -IoT-Fog-Cloud Network topology with a RAS in the IoT-Fog gateways. -Designing a Queuing Algorithm in the RAS -Identify Workflow between IoT, fog and cloud layers. -Identify Limitations.	-Developed a queueing and network model for resource allocation. -Refined the Architecture -Generated Deployment-Scripts. - A Scientific Paper Based on the Proposed Resource Provisioning Strategy RAS.
Physical Network Design	Implementing and Checking the Feasibility of Logical Network Design. - Proof-of-Concept Testbed Setup in Node-RED -Configuring the IoT devices, Fog Nodes and Cloud Servers. -Configure RAS in IoT-Fog gateway. -Implementing the network topology in Node-RED hosted at a High-Performance Machine.	-Queueing and Networking model Implemented -Criteria Decision Making Process.
Testing, Optimizing and Documentation Design	Evaluation -Investigate Suitability of The Queuing Model in reducing Queuing Time and Offloading Time Metrics. -Evaluate how RAS reduces Queuing Time and Offloading Time in Fog Computing Framework -Identify the Limitations of Experiments and Iterate Back to Design. -Compare to Base. -Do 1000 runs and collect results, average the results. -Write a thesis.	-Performed Evaluations of RAS in the Fog Computing Framework -RAS Reduced Queuing Time, Offloading Time and Improved Throughput -Presented the results on line and bar graph -A written thesis. -A scientific paper based on results.

The four phases can be categorized as either theoretical (*requirement analysis*) or technical (*logical network design, physical network design; and testing, optimizing*) as illustrated in **Figure 3-1**.

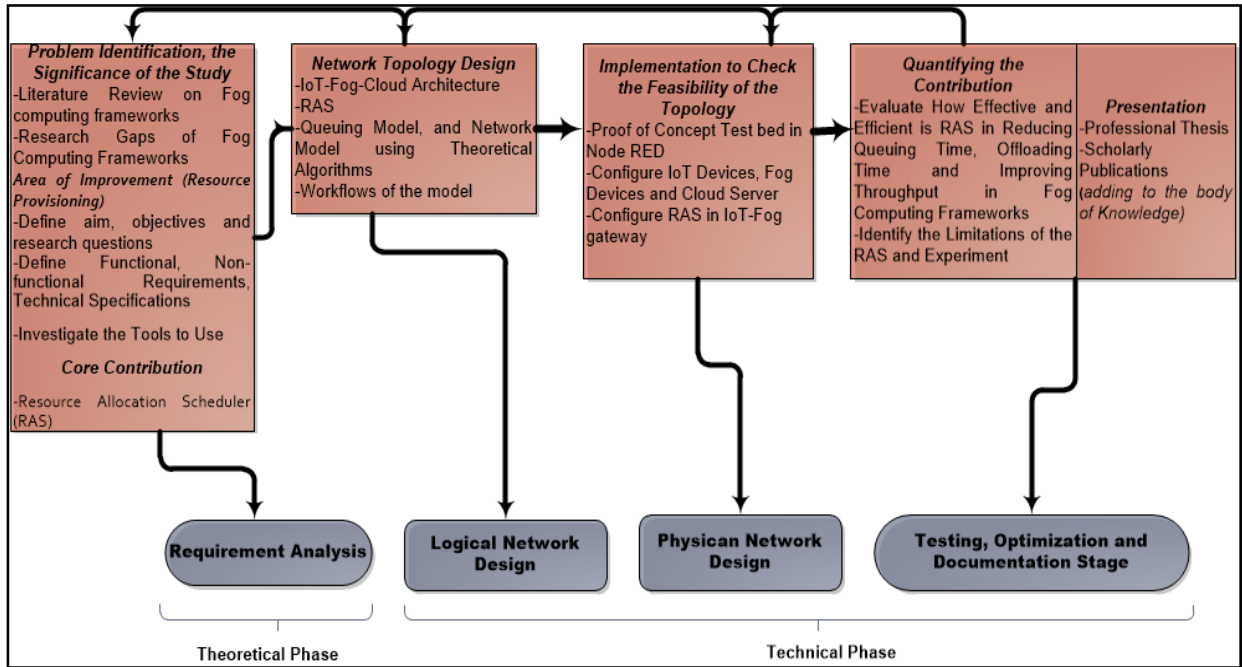


Figure 3-1: Top-Down Research Methodology phases

The following section describes the top-down approach phases in the context of this study.

3.2.1 Requirement Analysis

To first get “the big picture” of our research, systematic scrutiny of literature was done. We focused on cloud computing, fog computing and IoT to establish the research background for the study. Most of the reviewed literature which was scrutinized for IoT, fog computing and cloud computing was confined to between 2010 to 2020, which is a ten-year term. The reason for reviewing most of the literature in the ten-year range is because reviewing older literature, especially in computers, gives the wrong impression as far as identifying research gaps is concerned. It is a fact that in more than five years, technology would have changed significantly. The literature used was retrieved from electronic academic databases which included publications and journals. We mainly used SCOPUS, IEEE, Research Gate, Science Direct, International Journal for Fog Computing and Association of Computing Machinery

(ACM) digital libraries. These databases were searched using specific search keywords such as; “*Internet of Things, Fog Computing AND QoS*”.

It was from that scrutiny of the literature that the study identified challenges of fog computing in supporting IoT ecosystems as far as latency and QoS is concerned. We then looked at works that implemented fog computing in existing systems to address latency challenges and improve QoS. Furthermore, scrutiny on literature that microscopically focuses on providing strategies to keep improving latency and QoS in fog computing itself was done. In particular, strategy in fog computing that focused on improving reliability (service continuity and network quality), performance (application/task performance and service performance) and cost (energy efficiency, server operation cost and pricing of fog services) in fog computing frameworks.

It was from the literature review that we identified the different methods and strategies which were used in fog computing framework to minimize transit delay, improve availability, throughput and giving priority which are all QoS aspects. The main strategies used in most scrutinized literature included task offloading, service placement, resource provisioning and load balancing.

As part of the requirements analysis, the study managed to identify open research gaps which still exist in fog computing frameworks as far as reducing latency and improving QoS are concerned. The identified gaps included fog orchestration problem, computing problem and resource management problem. Those gaps still needs both the academia and the industry to research for solutions. The deliverable from the requirements analysis of this study was a journal paper “*A Review of Quality of Service in Fog Computing for the Internet of Things*” was published in ***International Journal for Fog Computing, 3 (1)***.. Most of that research paper's contents make up this study's chapter two. The full paper is attached in this thesis as **Annex 1**.

On the technical requirements and specifications, the analysis of existing frameworks on the reviewed scholarly literature helped in identifying the major functional and non-functional requirements for the fog framework.

a) Functional Requirements

The functional requirements consisted of the general functionality of the overall fog computing framework. The requirements included cloud management requirements in the cloud layer, fog node management requirements in the fog layer and RAS management requirements in the IoT-Fog gateway. Below is a brief explanation of each functional requirement and a brief explanation of its functions.

i. Cloud Management

Cloud management functions included parent identification, cloud resource provisioning, service placement, task request execution and service data storage. These were linked as follows:

- The cloud middleware identified and determined a resource in the cloud servers, in this case Front End for Node-RED (FRED), that would offer a service to a task request from the RAS and sent back a connection data signal.
- This prompted VMs and containers to be deployed and released in FRED to match the task requirements.
- RAS then sent the task to the deployed containers in FRED.
- Upon task arrival, the execution of the task by the containers in FRED began, and results were sent back to RAS. At the same time, the propagated service data was stored for further analysis.

ii. Fog Node Management

Fog node functions include device identification and creation of the framework topology.

- Fog nodes were created using Node.js and uniquely identified by an IP assigned to each fog node.
- A digital topology was created, and this allowed fog nodes to communicate with each other using cable or wireless technology such as Bluetooth and NFC.
- Fog nodes were periodically pinged to check if they were connected.

iii. RAS Management

RAS was hosted in the IoT-Fog gateways, and it was responsible for handling tasks request, deployed the services and managed the network. To be able to do this, a

“*reasoner, propagation component, watchdog and shared storage*” components were configured in the RAS. In short RAS “orchestrated, allocated and deallocated” tasks to and from fog nodes resources or cloud resources and monitored the resources. All the configured components would communicate, register and deploy services. An in-depth explanation for the functional requirements is found in chapter four of this thesis.

b) Non-Functional Requirements

For the proposed framework to execute, in case of device addition/failure, device accident or overload situations, it had to be guided by non-functional requirements. In the context of this research, we adhered to the following:

i. Scalability

For the framework to adapt to the ever-changing resource demands by the users, the CPU utilization for the framework was scaled to stay at less or equal to 80% for resource provisioning, and service placement. In comparison, the remaining 20% was reserved for the host device. Furthermore, RAS effectively and efficiently helped in the utilization of the fog resources by assigning time-sensitive tasks to the fog layer and sent non-time-sensitive tasks to the cloud. This helped to cut cloud cost as all time-sensitive tasks produced by IoT devices were sent to be processed in the fog layer. Moreover, this distinction where time-sensitive and non-time-sensitive tasks were to be processed assisted in achieving fast service deployment.

ii. Extensibility

The framework was designed using open-source software, the Node.js and Front End for Node-RED (FRED). The Node.js and FRED were selected because they are easier to modify whenever there is a need to do so. Furthermore, the framework was loosely coupled, and the APIs were clearly defined between diverse components. Also, a modular Node.js component structure was used to specify APIs which helped in meeting the precise requirements concerning our framework.

iii. Portability

In the development of the fog framework, Node.js was used for creating IoT nodes, fog nodes and FRED created the cloud server because they are light and could be easily used across diverse system environment. The open-source software used was platform-independent and could be quickly deployed and migrated.

c) Technical

The high-performance computer where the framework was hosted had 1100 terabyte (TB) storage capacity, 135 cluster nodes with 2900 processor cores and 11TB memory. Using a high-performance computer was essential to support the cloud CPU frequency, which was set at 10×10^9 cycles per second and memory capacity was 64 Gig. The fog node CPU frequency was set at 5×10^9 cycles per second, and memory capacity was 512 Megabytes. The IoT device CPU frequency was set at 6×10^6 cycles per second, and memory capacity was 128 Megabytes. The following were configured on this high-performance computer that is: 20 IoT devices, three IoT gateways, 10 IoT fog nodes and two cloud data centres. The IoT devices would produce at most 100 tasks. The network traffic to pass through the network was set and configured. The bandwidth was set at 20 Mega Hertz.

3.2.2 Logical Network Design

This section gives an overview of how the network topology was designed.

Since the study's main goal was not to develop a new architecture but to improve the architecture by adding RAS, the study adopted Skarlat *et al.* (2016) network topology. Skarlat *et al.* (2016) network topology is a well-known architecture in this field as it supports resource provisioning in both fog computing and cloud computing (Skarlat *et al.*, 2016). **Figure 3-2** shows the network design topology that was adopted.

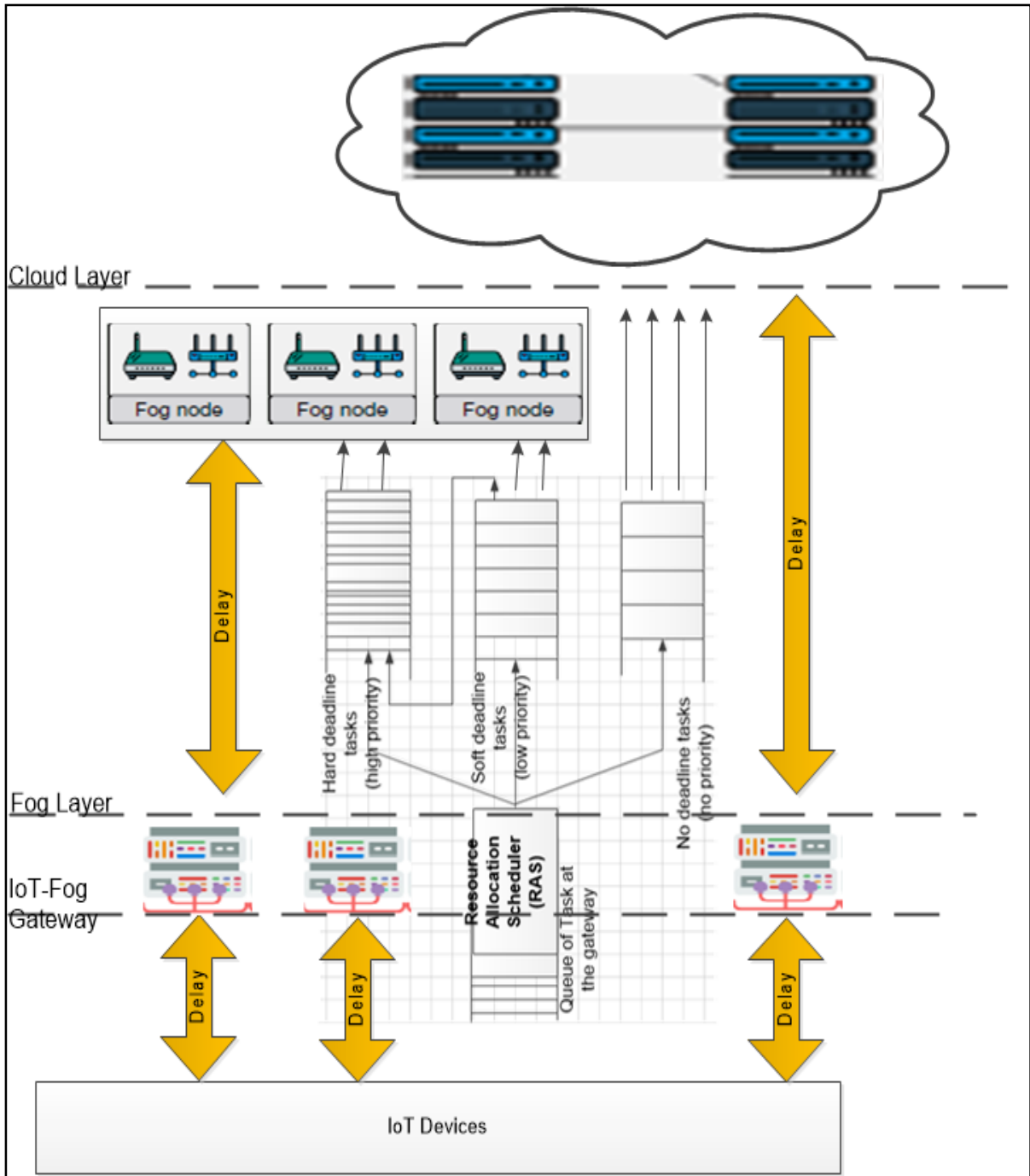


Figure 3-2: Network Design Topology

The network topology in **Figure 3-2** is made up of cloud layer, fog nodes, IoT-Fog gateway and IoT devices.

The cloud layer is the top level of this network topology. FRED, which was supported by OpenStack, was the cloud environment whose responsibility was to process non-time-sensitive tasks sent by the RAS. It also stored all the data of the topology for

future use. 192.168.1.101:8081 was used as the IP address and port for the cloud, respectively.

The fog layer was hosted and deployed on the Raspberry Pis. Three Logitech Media Server with IP address and port 192.168.1.102:8080; three Logitech Harmony Hub with IP address and port 192.168.1.103:8080; and four Samsung SyncThru Printers with IP address and port 192.168.1.104:8080 were used as the fog nodes and connected to the Raspberry Pis. As shown in the diagram, fog nodes were directly connected to the cloud layer so that they would be able to send data that needed storage for future use.

The IoT-Fog gateways, in this case, Linksys wireless AP, NETGEAR routers, Belkin WeMo Switches, Xiaomi gateway were also hosted and deployed on the Raspberry Pis. As shown in Figure 3-2, both the cloud server (FRED) and fog nodes were directly connected to IoT-Fog gateways where RAS was hosted. RAS controlled, orchestrated and supervised the cloud server (FRED) and the fog nodes which processed data from the connected IoT devices. The queuing algorithm in RAS helped in the assignment of resources to IoT task based on their priority.

In the edge layer, five temperature sensors, four humidity sensors, eight security cameras, one telldus live and two Sonos speakers were the IoT devices connected via home assistant to the respective Raspberry Pis. A sensor module was used to connect temperature sensors and humidity sensors to the Raspberry Pis.

To connect the cloud server, the fog nodes and every Raspberry Pi to the internet, a robust Linksys wireless AP was used to create a wireless LAN network. The reason for connecting every component to the internet was to allow the fog servers to download Docker Image data which always assisted in creating and deploying dynamic services.

From this phase and the technical requirements and specifications, a paper titled "*Resource Allocation Scheduler Strategy to Minimize Round-trip Time in Fog Computing*" was written, submitted and it is under review on the Journal of Computer Science.

3.2.3 Physical Network Design

Five temperature sensors, four humidity sensors, eight security cameras, one tellus live and two Sonos speakers were used to create a home set up. These devices were simulated as the IoT devices and connected directly to the simulated IoT-Gateways, which hosted the RAS. The simulated IoT-Gateways included Linksys wireless AP, NETGEAR routers, Belkin WeMo Switches, Xiaomi gateway. These gateways were linked and connected to the fog nodes Logitech Media Server with IP address and port 192.168.1.102:8080; Logitech Harmony Hub with IP address and port 192.168.1.103:8080; and Samsung SyncThru Printer with IP address and port 192.168.1.104:8080. The IoT-Fog gateways were also linked and connected to the cloud server FRED which was supported by OpenStack (192.168.1.101:8081).

A Node-RED cross-platform runtime simulation environment was used. The starting point was to make sure we created a proof of concept testbed in Node-RED consisting of the above mentioned IoT device, IoT-Fog gateways, fog devices and cloud server. The runtime environment supported Node.js and Front End for Node-RED (FRED) which were the building blocks. Node.js was used to the IoT devices and fog nodes; build back end services; the application programming interface (API). The reason why Node.js was used is that it is highly-scalable, data-intensive and used for real-time apps. Its non-blocking or asynchronous nature allows a single node to handle multiple requests. FRED was used as a cloud platform. Most tools in Node-RED were used to define and create nodes, sequences, and flows for the simulated smart home. In the progression of the design and implementation, several codes in Node.js were tailored to achieve a specific goal.

3.2.4 Testing and Evaluation

The starting point was to make sure that there was communication between the IoT device, RAS, fog nodes and cloud servers by starting the network. Every node was pinged periodically to make sure it was up and running before sending the tasks. The IoT devices generated tasks, sent them to the RAS to check if the framework was functioning according to what it was supposed to be doing. This was done for ten times before collecting results as to make sure that all connections and communications were perfect. The debug and dashboard in Node-RED showed the connection status, any errors and how frequent the data is being sent or received.

After making sure that everything was up and running, 1000 independent runs were done; results collected and averaged for each parameter which was tested. In particular, we were concerned about determining the round-trip time. To determine round-trip time, this study focuses on queueing time and offloading time parameters.

a) Queuing Time

In order to assess queueing time which is the time a task waits in the queue before it is assigned to a fog or cloud resource based on its priority category (high, low and no priority tasks); tasks were sent automatically from IoT devices at the same time in batches of 5 after every 10ms time-stamp. A maximum of 40 tasks was assigned for each of the three categories of the tasks. A total of 1000 independent runs were done; results collected and averaged for each parameter which was tested. The results for each task category were collected at the dashboard and recorded in milliseconds. The test aimed at establishing the effect of having more task and how RAS handles these tasks as far as resource allocation is concerned.

b) Offloading Time

To evaluate offloading time, which is defined as the time taken to upload, process and download a task from IoT devices to RAS to fog or cloud device based on its priority category (high, low and no priority tasks); tasks were sent automatically from IoT devices at the same time in batches of 5 after every 10ms time-stamp. A maximum of 40 tasks was sent for each of the three categories of the task. A total of 1000 independent runs were done; results collected and averaged for each parameter which was tested. The results for each task category were shown at the dashboard and recorded in seconds. The test aimed at establishing how RAS handles tasks as far as resource allocation is concerned. Moreover, to check if the introduction of RAS reduces overall offloading time.

c) Throughput

If queueing time and offloading time of tasks are reduced based on the set time-stamp set for high priority tasks, low priority task, then it is considered that throughput is improved throughput. The higher the number of tasks completing their processing, the higher the chances of getting improved throughput. In this study, throughput was calculated as the number of tasks that complete their process within a time-stamp

based on the arrival rate. Therefore, the throughput was determined from the results obtained from queueing time and offloading time. The evaluation was done to check whether the introduction of RAS brought about any advantages in improving throughput or not.

Based on this phase and other parts of this research, a paper titled “*A Fog Computing Framework for Quality of Service Optimisation in the Internet of Things (IoT) Ecosystem*” was submitted, accepted for a conference and the paper will be published in the IEEE digital library attached as **Annex 2**.

3.3 Conclusion

The main goal of this chapter was to explain, recount and summarise how the aim, objective and research question were addressed. The chapter explained how the top-down research methodology was used in the context of this research. In short, the chapter presents: i) the method adopted, techniques used and how they were used for answering the research questions, ii) why the methods are relevant to the study aim and objectives iii) an explanation how we used them and finally iv) how data was collected and presented. The chapter answered this study's research question: “which approaches can be used to build an adaptive resource allocation scheduler in fog computing framework for IoT environment?”

Chapter Four: Framework Design and Implementation

4.1 Introduction

The chapter serves to answer the fourth research question: “Can an adaptive resource allocation scheduler in fog computing framework, which is based on tasks requirements and priorities be successfully developed?”. The chapter articulates the framework design of this study and how it was implemented. The chapter is structured as follows: **Section 4.2** presents the framework design and explains how the components of the framework work, the aspects expected to be supported with the framework and workflows. To check for the feasibility of the framework, **Section 4.3** presents the implementation, which is guided by top-down system design as explained in chapter three. More emphasis will be on IoT services, testing and evaluation of the effectiveness of RAS considering round-trip time (queuing and offloading time). **Section 4.4** presents the synopsis of the whole chapter.

4.2 Framework Design

As can be seen in **Figure 4-1**, the framework comprises of the edge layer where IoT devices are found, IoT-Fog gateways where Resource Allocation Scheduler is hosted, the fog layer and the cloud layer. It is important to note that our main goal for this research was not design a new framework but to improve the existing fog computing framework by introducing the RAS. As such, the main emphasis is on Resource Allocation Scheduler and the fog layer. To get a better understanding, the edge layer, the fog and cloud layer functionalities are listed and concisely described. **Figure 4-1** below shows the cross-sectional design of the fog computing framework with a resource allocation scheduler.

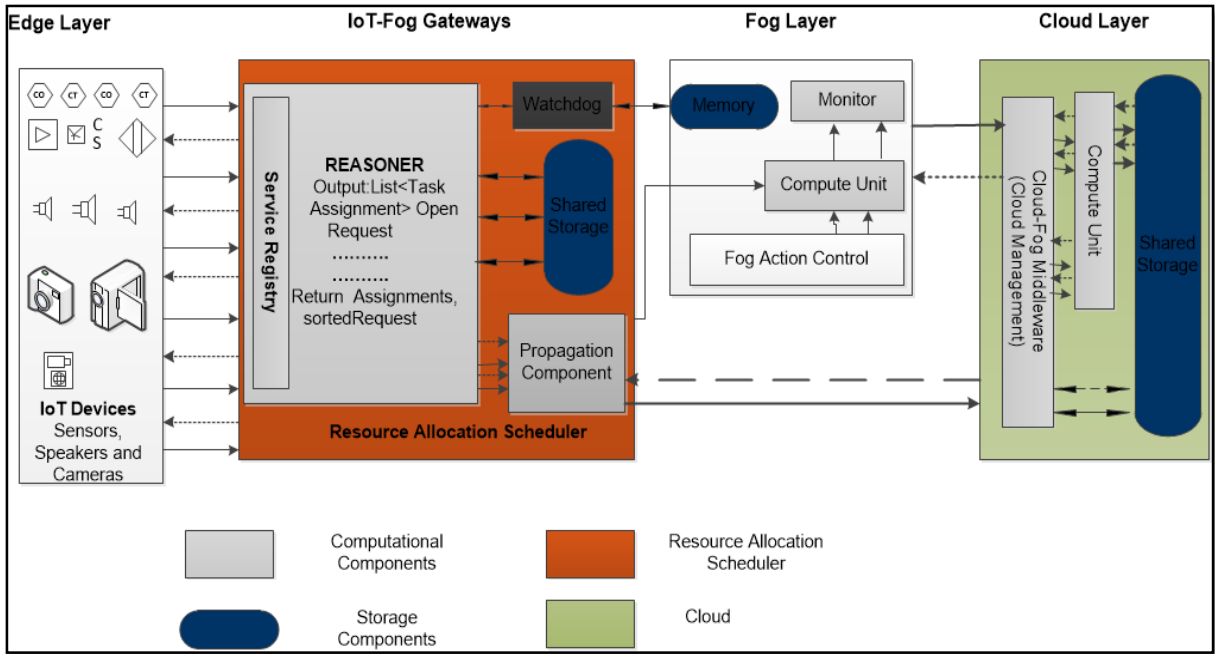


Figure 4-1: Cross-Sectional Design of the Fog Computing Framework

4.2.1 Components of the Framework

In this sub-section, the specific functions and behaviour of the resource allocation scheduler in fog computing and how it relate to IoT devices, fog layer and cloud layer are presented. Moreover, an indepth explanation of **Figure 4-1** is highlighted in the following sections.

4.2.1.1 Edge Layer (IoT devices)

The edge layer comprises of IoT devices that connect and communicate with other devices using NFC, RFID, Bluetooth, Wireless Sensor Networks, Wi-Fi, perform tasks or respond to events without explicit instructions. In the case scenario of a smart home , the application area under consideration, IoT devices can include laundry appliances, water treatment appliances, and water filtration systems, kitchen appliances, smart LED lighting, sensors and actuators. For this research, five temperature sensor, four humidity sensor, eight security cameras, one toldus live and two Sonos speakers were used. The IoT devices in the edge layers generate tasks that need to be processed. The IoT devices can compute and process tasks on their own. Tasks requiring more computational power and higher storage capacity, cannot be processed within the IoT devices themselves as those devices have low computational power and storage capabilities. Therefore, if some tasks demand more computational power, such tasks

are sent to the fog layer or cloud layer for processing via the resource allocation scheduler (RAS).

4.2.1.2 IoT-Fog Gateway (Resource Allocation Scheduler)

The IoT-Fog gateways host the introduced Resource Allocation Scheduler (RAS). For this research, the IoT-Fog gateways included Linksys wireless AP, NETGEAR routers, Belkin WeMo Switches, Xiaomi gateway. Generally, the RAS responsibility is to orchestrate, allocate, deallocate, reallocate and monitor tasks from IoT to fog layer and cloud layer. When responses (processed tasks) come back from either the cloud or the fog layer to the IoT, they pass through the RAS which would assign the response to the correct IoT device that have sent the task. The RAS contains the following components; the *reasoner*, the *propagation component*, the *watchdog* and the *shared storage*, which assist in resource allocation. The functionalities of these components are explained below.

a) Reasoner

When a task is sent from any IoT device to the RAS, the *service registry* marks the task based on which IoT device it came from. It is the responsibility of the *reasoner* to do resource allocation for the entire framework and make decisions to either send a task to the fog node or cloud based on the reasoning shown in the flow chart **Figure 4-2**.

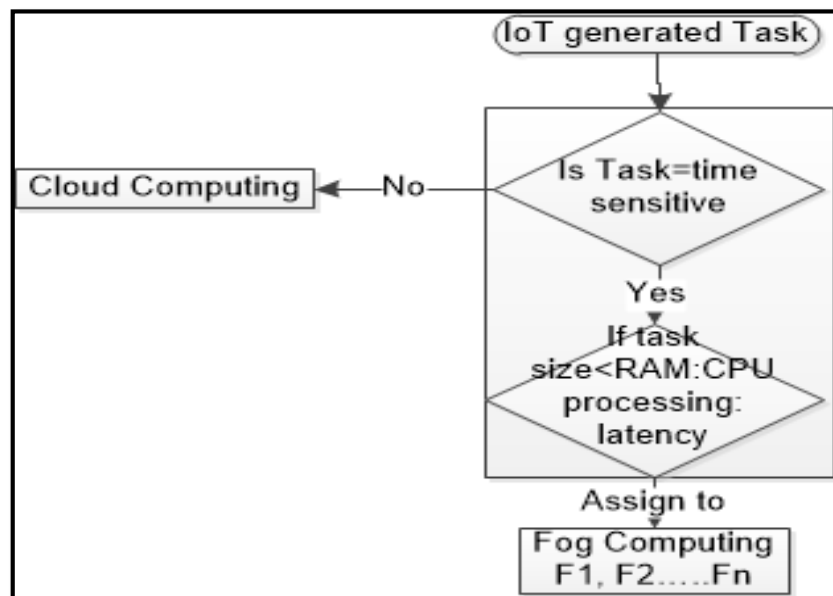


Figure 4-2: Decision Making Flowchart

When a task arrives at the reasoner, the reasoner checks if the task is time-sensitive or not. If the task is not time-sensitive, it is sent directly to the cloud. If the task is time-sensitive, some decisions are made and it is sent to the fog layer.

The algorithm in the reasoner helps to choose the correct fog node for a specific task based on the QoS requirements of that task. Factors such as distance and the processing power of the fog nodes are considered as they play a pivotal role in time-critical tasks, as explained in previous sections. Moreover, watchdog events are considered by the reasoner, as they help the reasoner to be more effective when making decisions. If there is a fault or errors at the fog node, events are triggered, and signals are sent to the reasoner. The reasoner assigns the task that would have been processed in the fault node to the next available and capable fog node. The reasoner uses a modified first come first serve (FCFS) heuristic approach **Figure 4-3** and applies the following rules in the queue.

```

Output: List<TaskAssignment> assignments, List<TaskRequest>
openRequests
// init fields
1 assignments ← [];
2 round = 0;
// sort children and task requests according to the service type
3 sortedRequests ← sortByServiceType(requests);
4 sortedChildren ← sortByServiceType(children);
// loop over children, service types, and task requests
5 for child ∈ sortedChildren do
6   for serviceType ∈ child.serviceTypes do
7     for request ∈ sortedRequests do
8       if serviceType == request.serviceType then
9         utilization ← getUtilization(child);
10        containers ← getContainerCount(child);
11        if checkRules(utilization) and
           containers < MAX_CONTAINERS then
12          // deploy container and save the assignment
           container ← sendDeploymentRequest(child, request);
13          assignments.add(child, request, container);
14          sortedRequests.remove(request);
15        end
16      end
17    end
18  end
// check if the iterator is finished, the max rounds is not yet
reached, and not all requests are assigned
19 if !sortedChildren.hasNext() and round < ROUNDS
   and sortedRequests.size() > 0 then
20   // increase the round counter and reinitialize the children iterator
   round = round + 1;
21   sortedChildren.reStart();
22 end
23 end
// return the assignments and the still open requests
24 return assignments, sortedRequests

```

Figure 4-3: First Fit Algorithm in RAS for Resource Provisioning

In the reasoner, the algorithm considers fog nodes in the fog layer and IoT device task requests as input. A round counter field and the needed assignments are initiated in the line 1 and 2, respectively. The IoT device task requests and fog nodes are sorted based on their service type in the line 3 and 4. This is done to improve loop performance and deployment time. The loop over of fog nodes, followed by fog node's service types and finally the requests are made in the line 5 to 7. Line 8 is responsible for making sure that the fog node has assigned tasks that suit its service type. It is vital to repeatedly re-evaluate the RAM, storage and CPU utilization and the amount of already deployed containers to avoid overloading the fog node, which will compromise its performance and affect the quality of service.

To make sure that the fog node is not overloaded the fog node, in the line 9 and 10 requests RAM, storage and CPU utilization from the fog nodes. This supports the watchdog in the line 11 to monitor that the CPU<80% which are the predefined rules, thus avoiding exceeding the maximum number of containers to be deployed. If the fog node still hosts another service, it will send a detailed request of the processing space it still has to RAS.

The fog node sends a message if it can no longer accept any task request. It is the responsibility of the storage and watchdog in the RAS to keep track and store the information of the tasks that have been deployed and where they have been deployed; the pending tasks to be deployed, and the information of the fog nodes that are still free. At the end the algorithm returns all this information. Tasks requests that have successfully been executed are removed in the input section to avoid redeploying the same task request many times.

The above explanation will continue until a fog node is looped through and is finished. Line 19 is executed if and only if the last fog node loop is completed, the round counter is smaller than the maximum defined rounds and if there still exist unhandled task request.

- Rules of Queuing

Figure 4-4. below highlights the queuing model for simplicity.

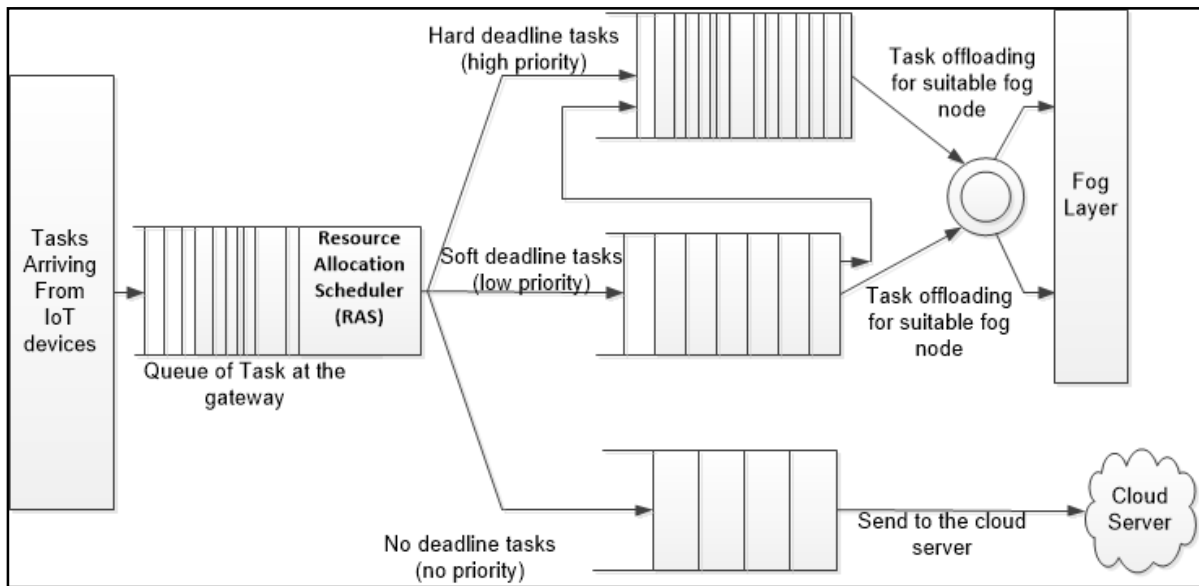


Figure 4-4: Priority queueing model for IoT gateway.

The *reasoner* in the RAS, which is in the gateways (G), receives multiple numbers of tasks from different IoT devices at the edge layer that needs to be assigned to either fog nodes or cloud servers. In the reasoner there will be a time-slotted system denoted by $ts=\{1,2,3,\dots\dots n\}$ and the time slot is denoted by AT . When there is no task to be assigned in the reasoner, the queue denoted by Q will be empty, which means when $Q=\emptyset$ then $ts<0$. The task will be arranged using the First-Come-First-Serve (FCFS)/ Q concept where Q represents the size of the queue. Using the Poisson process, it is considered that the time interval of arrival between successive task is exponentially distributed. There are two things to be considered, that is (a) the arrival rate (ar) of the task and (b) the service rate (sr) of the computing device that is hosting the RAS. These two determine how the queue will move. Above and beyond arrival rate and service rate, the moving of the queue is also affected by whether the computing devices in the fog layer or cloud layer are free or not at a certain time-stamp. The RAS will classify the tasks into three main categories, namely time-sensitive task (high priority tasks), low time-sensitive (low priority tasks) and not time-sensitive (no priority tasks).

As explained in the earlier chapters, no priority tasks are tasks that are not time-sensitive and they do not have any stipulated time to be processed. Contrariwise, “high priority” tasks are time-sensitive tasks which should be processed within a specific time. If not processed, the task will no longer be valid for the IoT device. Such tasks are time and latency-sensitive task. In almost a similar fashion with “high priority” tasks, “low priority” tasks are tasks whose processed output is valid up to a certain extent, if that time is not met, some penalties will be applied, but it will wait to be processed, and the IoT device will use that output even though the output will have failed to meet their corresponding deadlines. All these tasks would be placed in three different queues denoted by $Q=\{1,2,3\}$. $Q1$ will be for high priority tasks, $Q2$ for low priority tasks and $Q3$ for no priority tasks.

Even though the tasks in $Q2$ are not very time-sensitive, they should not suffer a starvation problem. The starvation problem occurs when $Q1$ tasks keep on coming, which will result in $Q2$ tasks not to be processed. Therefore, to avoid the starvation problem, after 10 seconds time-stamp, $Q2$ tasks that are in the queue for a defined time without being assigned to any computing resource will be promoted to $Q1$ which is of a higher priority. Those in $Q3$ are sent directly to the cloud since they are not time-sensitive.

As shown in the diagram **Figure 4-4**, in the queuing of tasks, three queues are formed, as explained above, the tasks would be placed in three different queues denoted by $Q= \{1,2,3\}$. $Q1$ would be for high priority tasks, $Q2$ for low priority tasks and $Q3$ for no priority tasks. The algorithm in the RAS would consider the following rules.

- i. In the event that the queue of high priority tasks is not empty ($Q1 \neq \emptyset$), then the tasks in that queue are scheduled using the first come first serve order in the RAS for offloading to the available fog layer. If the first task requirements do not match the available resource, the next task in the queue is considered and assigned to the available fog resource. This is done not to waste resources and waste the time for tasks in the queue that can match and utilise the available resources.
- ii. If the queue of high priority task is empty ($Q1=\emptyset$) and the queue with low priority tasks has some tasks ($Q2 \neq \emptyset$ and $Q2>0$), then the tasks in that queue are given priority and scheduled using the first come first serve order in the RAS for offloading to the available fog layer. If the first task requirements do not match

the available resource, the next task in the queue is considered and assigned to the available fog resource. This is done not to waste resources and waste the time for tasks in the queue that can match and utilise the available resources.

- iii. Then if the queue of higher priority tasks has tasks ($Q1 \neq \emptyset$ and $Q1 > 0$) and the queue with lower priority tasks has tasks ($Q2 \neq \emptyset$ and $Q2 > 0$), $Q1$ tasks have to be processed first while the first task in the $Q2$ queue is placed at the end of $Q1$.
- iv. When both queues with high priority and low priority tasks are empty ($Q1 = \emptyset$ and $Q2 = \emptyset$), then the queue with no priority tasks will be given all the priority in terms of networking and message routing resources and are scheduled using the first come first serve order in the RAS for offloading.
- v. When all the three queues, high priority tasks queue, low priority tasks queue and no priority tasks queue denoted by ($Q1 \neq \emptyset$ and $Q1 > 0$), ($Q2 \neq \emptyset$ and $Q2 > 0$), ($Q3 \neq \emptyset$ and $Q3 > 0$) respectively, are not empty then RAS will assign tasks based on priority. After time-stamping $Q1$ and $Q2$ tasks, it will send all $Q3$ tasks to the cloud for scheduling as to create space in the holding memories of the RAS.
- vi. When the high priority task queue has tasks ($Q1 \neq \emptyset$ and $Q1 > 0$), the low priority task queue is empty ($Q2 = \emptyset$) and the queue with no priority task has tasks ($Q3 \neq \emptyset$ and $Q3 > 0$), the RAS will time-stamp all the high priority tasks, then send all the $Q3$ tasks to the cloud.
- vii. Finally, when all the queues don't have anything ($Q1 = \emptyset$, $Q2 = \emptyset$, and $Q3 = \emptyset$), then the fog layer and cloud layer have to be put in a sleep mode to save energy and the cost of using these two layers as they will be idle.

Based on the above analysis of queuing rules, the main goal of this research would be considered achieved if the round-trip time is reduced, especially for those tasks with deadlines. With the priority approach in the RAS, queueing time, task uploading and downloading time should be minimized, which leads to an increase in the number of tasks meeting their deadline.

b) Watchdog

The *watchdog* persistently monitors the load given to each fog node and also monitors performance to avoid overloading fog nodes at the fog layer. Overloading fog nodes

can compromise QoS and round-trip time. Furthermore, the watchdog monitors faults or errors in the fog layer. If there is fault or errors at the fog node, events are triggered, and signals are sent to the reasoner. The reasoner then re-assigns the task that would have been processed in the fault node to the next available and capable fog node. These *watchdog* events are also put into considerations by the reasoner, as they help the reasoner to be more effective when making decisions.

c) Propagation Component

After the reasoner has decided to either send the task to the fog layer or the cloud layer based on the task priority, the task is forwarded to the propagating component. It is the responsibility of the *propagating component* in the RAS to propagate task request and service data to the fog node or cloud-based on the decision made by the reasoner.

d) Shared Memory

The purpose of *shared memory* in RAS is to hold service registry information and all the information of fog nodes and IoT devices that are registered in the network. Each fog node and IoT device are given unique identities, which would help to assign the correct response to the proper IoT devices.

As can be visualized in **Figure 4-1**, the RAS has a memory. This memory is shared and is made up of *Redis FCN* and *Redis Shared*, which are database containers, as shown in **Figure 4-5** below. *Redis FCN*'s responsibility is to hold data that is local such as device utilization which includes RAM, CPU, and storage. Whilst *Redis Shared* hold the Docker images of the services that would have been deployed in the fog layer and cloud layer. The two highlighted databases are saved separately as to maintain and preserve flexibility and replaceability.

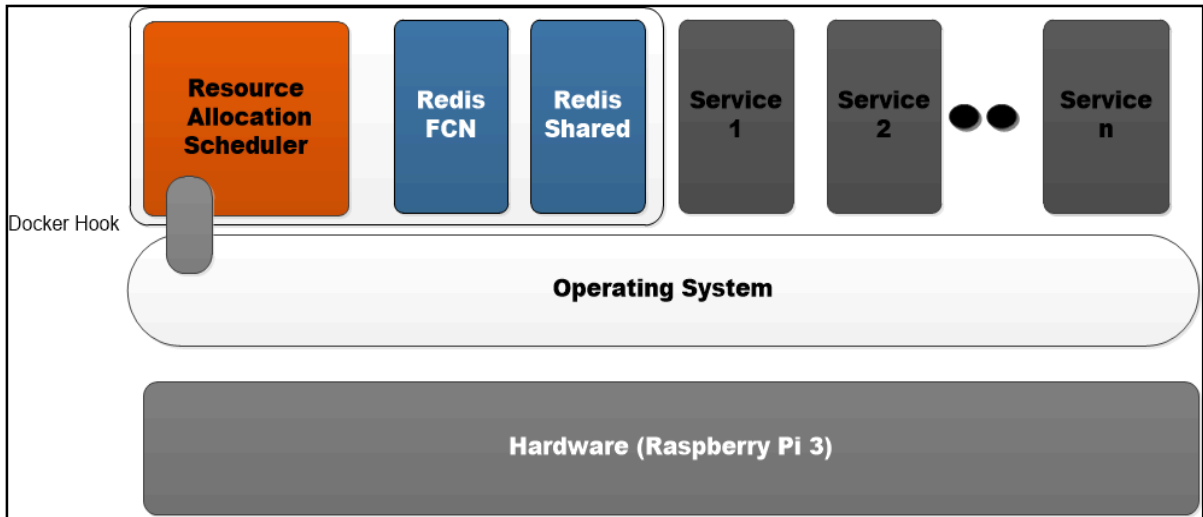


Figure 4-5: Reasoner, Redis FCN, Redis Shared Deployed on the Raspberry Pi

The functions of the Resource Allocation Scheduler (RAS) are summarized below:

- i. Register all fog nodes that are both new and old, the cloud and monitor the status. Calculate a new resource provisioning plan according to events that would have encountered device failure and device overload. Store all the existing fog nodes and cloud processing capabilities, which include processor, speed, RAM, storage space.
- ii. Receives task requests from the IoT devices (edge layer) and check for task request in the network. Receives task request that needs to be processed, label the task request, execute the incoming task request and propagate them to the responsible fog nodes in the fog layer or cloud depending on the tasks requirements.
- iii. Analyze, monitor data and fire events when the previously specified QoS threshold is exceeded. If the limit is exceeded, the allocator has to make a call to either migrate that task to another capable fog node.

4.2.1.3 Fog Layer

The fog layer is made up of any network resources that include; mobile stations, servers, switches, routers, and so on, depending on the area of application. In this study, Logitech Media Server, Logitech Harmony Hub, and Samsung SyncThru Printer were used as fog layer nodes. These network resources offer their resources to add computation capabilities, pre-processing and temporary storage within the network

and are named fog nodes. Because of their proximity to the ground and having the characteristics as described in the background **Section 2.4.1**, they provide lower latency compared to cloud computing which results in offering improved QoS. Fog nodes will receive time-sensitive tasks from RAS and execute them before sending the response back to the RAS and or to the cloud for the information that requires long-term storage. A fog node has four parts which include fog action control, compute unit, monitor and memory.

- i. Fog Action Control*-This part controls all the action that happens in the fog nodes. It communicates with the propagation component of the reasoner.
- ii. Compute Unit*-This is the central processing unit (CPU) responsible for computing and processing of tasks.
- iii. Monitor*-This part of the fog node monitors the performance of the CPU and other components of the fog node. It always communicates with the watchdog based on the fog node status.
- iv. Memory*-This part stores the fog node details and all the actions or commands that need temporary storage during processing.

- *Service Deployed in Fog Layer Nodes*

If the task request is time-sensitive, it is deployed by the propagation component in RAS to the the fog layer. The fog layer has different fog nodes where the task request is distributed depending on the task request requirement. The fog nodes are presumed to be administered on Raspberry Pi, ARM processor architecture and deployed in Docker Containers as shown in **Figure 4-6**. In the Raspberry Pi, where each fog node would be running, there would be a software called Hypriot, which acts as an OS. The Hypriot's responsibility is to run a Docker runtime, which enables deployment of services in the Docker Containers. A specific Docker hook in the Docker Containers host allows applications in the fog node to start and stop containers that exist in the same OS using the same Docker Runtime. Moreover, fog nodes also have Redis FC, which acts as a database (memory) to persist and read device utilization of the RAM, CPU, and storage, which is the local needed data.

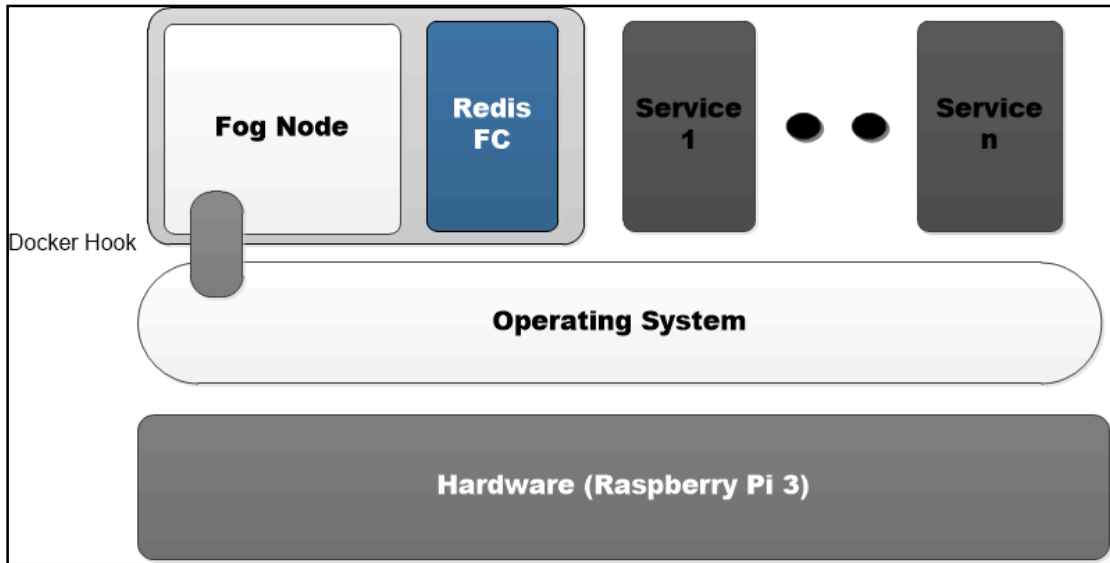


Figure 4-6: Fog Node Deployment on the Raspberry Pi

4.2.1.4 Cloud Layer

If the task request is not time-sensitive, and it is CPU intensive, the Resource Allocation Scheduler would deploy task in the cloud layer, which supports CPU intensive tasks. It is important to note that there are many kinds of research done for resource provisioning in the cloud, as indicated by the surveys (Zhan *et al.*, 2015), (Singh and Chana, 2015). As such, it was not our focus. However, for clarity sake, this is what happens in cloud service deployment.

In the cloud, services are deployed in dynamic VMs, which are based on an Intel processor architecture (Jim, 2014). According to dynamic cloud resource demand, these VMs should always be deployed, managed and stopped. When the Resource Allocation Scheduler deploys a task request to the cloud, it passes through the cloud-fog middleware. This cloud-fog middleware would start a new VM which is based on light-weight CoreOS operating system running a Docker environment. The VM would continue running until it is filled with Docker Containers. If the VM is now full that it will not accommodate another container, this triggers the creation of a new VM again, and the process goes on. In case the containers are stopped, which results in VM having no containers, the VM would be released to save cloud resources.

In the event that two task requests are deployed in the cloud-fog middleware by the RAS, the cloud-fog middleware would start a new VM and deploy the first requested service container. When the first service is deployed successfully, the signal is sent to

the RAS and the cloud-fog middleware will then deploy the second request in the same VM, provided that it has enough resources, or else a new VM is started. When both services are finished, the VM is stopped by the cloud service and the resources are released.

The above can be summarised as follows:

- i. The cloud management determines the closest cloud server to a requesting device and sends back appropriate connection data to the RAS.
- ii. It then deploys and releases resources based on the resource demands of the tasks send by RAS.
- iii. Incoming task requests are handled by deploying memory on running cloud servers.
- iv. The task requests are then executed in the resources provided by the cloud servers, send back the task response and store the details in service data storages for further analysis.

4.2.2 Aspects Expected to be Supported with the Framework

The framework should support communication following the principles adopted from (Pautasso *et al.*, 2008):

- i. *Statelessness*: the communication of components is independent of one another, and the components should not store any state.
- ii. *Uniform Interface*: When executing, the components follow GET, POST, PUT and DELETE commands.
- iii. *Resource Identification*- When commands are being executed, specific URIs are mapped to the component's execution methods on the APIs.
- iv. *Self-descriptive messages*- All messages that are exchanged in form and content are separable and should be self-descriptive as such.

4.2.3 Workflows

Sequence diagrams are used to explain the essential workflows of the framework. In these sequence diagrams, vertical direction envisages time, whereas horizontal direction envisages the communication between different components. Three things that happen in this framework are: new fog nodes can be registered or deregistered, and the task is processed.

a) Pairing and Service Deployment

Figure 4-7 shows the pairing of the device and subsequent service deployment.

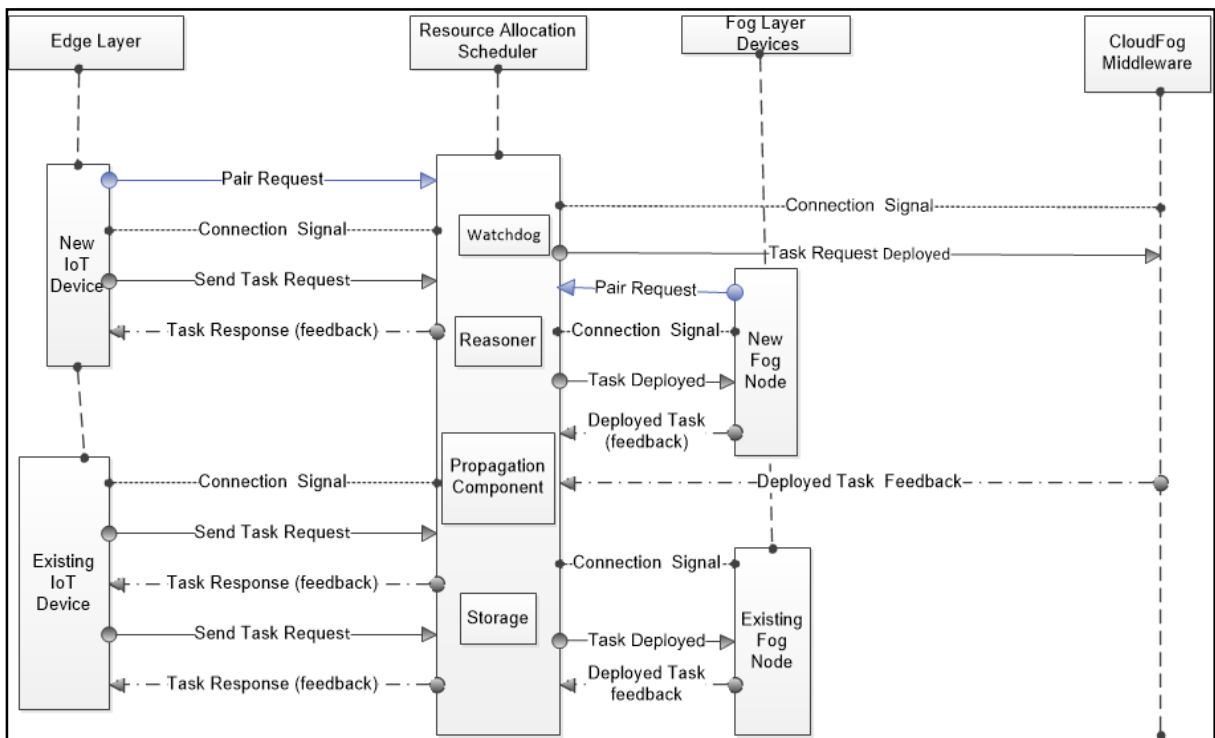


Figure 4-7: Pairing of Fog Nodes, RSA, IoT Devices and Service Deployment

When a new fog device joins the fog layer to avail its resources, it sends a signal to the RAS to be registered. The RAS would then register all the details, which include the device's processing power and RAM size. The device is then instantiated as a fog node. Once a new fog node is registered, there is no need to register its details again because these would have been stored. Assuming that there is pending task requesting for a resource and the newly installed fog node does have the required resource requirements that match the task, RAS would immediately deploy the task. Once the service is deployed, the newly added fog node would be able to read and execute the task and return the response to the RAS. The same happens when new IoT devices such as sensors, actuators, laptops, smart television, join the network. They get registered at the RAS as new IoT device, and the type of data they send is also recorded. This is done so that the RAS would keep that information to avoid repeating the process of identifying the type of data sent by the device each time it sends the data, thus minimizing future delays.

b) Resource Allocation Scheduler Assignment of Task

When a new task is sent from the IoT device, it goes through the RAS where it is labelled whether it is of high priority or not, specifying its QoS matrices requirements. If the task request is time-sensitive, it is sent to the fog layer, and if it is not time-sensitive, it is sent to the cloud layer for processing. If the decision by the RAS is to send the task request to the fog layer, the RAS will choose the most fitting fog node to deploy the task request. After the RAS do the above reasoning, it then deploys the task to the fog node. If the service is successfully deployed, the fog node sends a signal back to the RAS for monitoring purposes and the fog node starts immediately executing the deployed task. After the task has been executed in the fog node, the response is sent back to the RAS, which further forwards it to the specific IoT device. A copy of the response and other processed details are sent to the cloud for long term storage. If the task is not time-sensitive and requires more computational power, the RAS would flag it as such and deploy it to the cloud. In the cloud, it is assigned to the virtual machines (VMs) which process the request. The component, which would handle the task, would send a signal to the RAS as an indication that it was deployed successfully and for evaluation purposes too. **Figure 4-8** is a diagrammatic representation of what takes place at this stage.

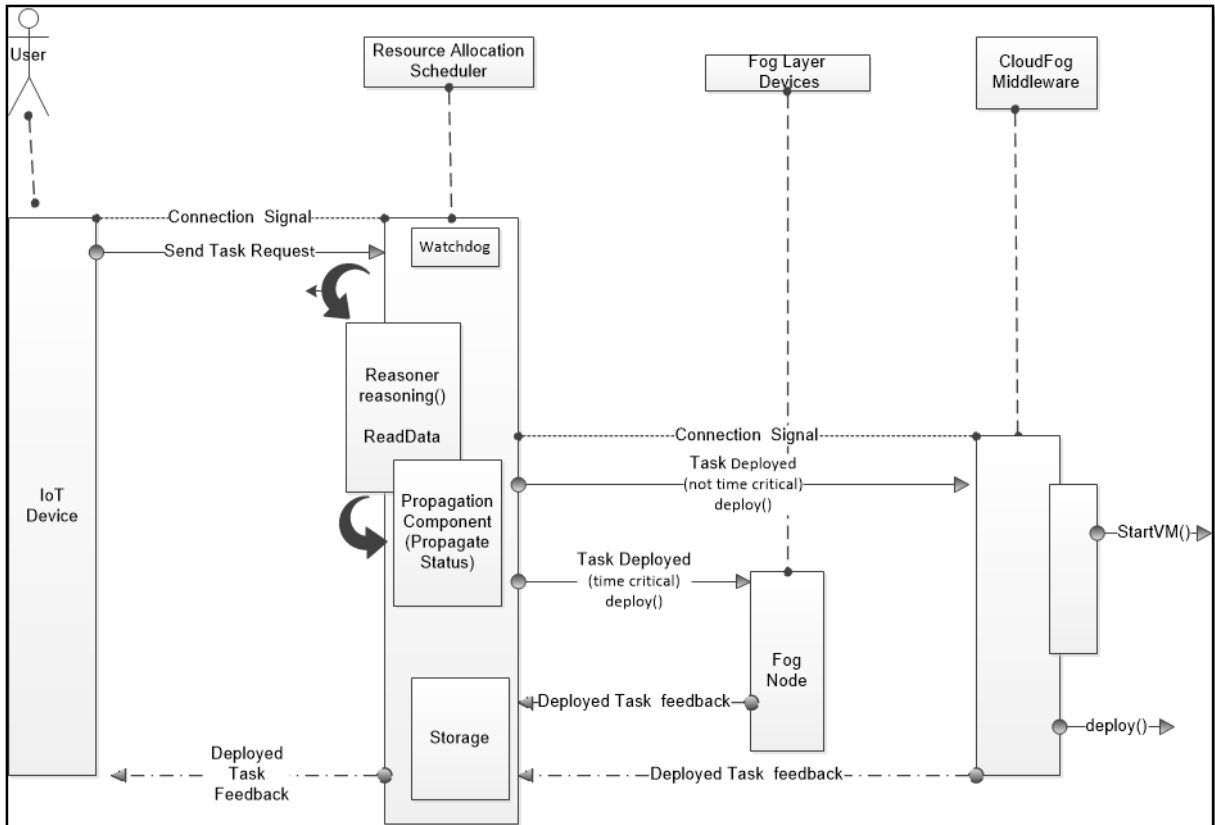


Figure 4-8: RAS Assignment of Task and Processing

As explained above, two cases can prompt re-planning, and this is how they are implemented.

Case 1: When a fog node leaves the fog layer, whether it was a fault or the owner decided to withdraw the services, the RAS should check whether the fog node had a task that it was processing. The reasoner will have to deploy the task to available fog nodes if there are any. If there are no available fog nodes, the task has to be put at the beginning of the queue, thus giving them the highest priority.

Case 2: When a new fog node joins the fog layer, the task at the queue that suits in the resource would be given priority. This selection is done by considering the first task at the queue following the order at which the tasks has arrived until the correct task is found that matches the CPU and RAM in that new fog node.

4.2.4 Application Programming Interfaces (APIs) Endpoints

The most vital APIs endpoints are presented in this section. These include endpoints between a task request IoT device and RAS, and between a deployed service on fog

node and the RAS, RAS and the cloud-fog middleware. These can be seen in **Table 4-1** to **Table 4-5**.

Table 4-1: Register Service Endpoint

<i>Register Service</i>	
Description	<i>This endpoint enables users to register services in the shared service storage, located on Resource Allocation Scheduler, for future deployment.</i>
<i>Request</i>	
URL	<i>POST http://fcn1:8080/shareddb/register</i>
JSON	<i>{ "serviceKey": "busy-image", "dockerfile": "FROM rpiBusybox\n ...", "volumes": "/usr/lib/...:/usr/lib/...", "exposedPorts": ["8105"], "privileged": true }</i>
<i>Response</i>	
Description	<i>Returns a status flag if the registration was performed successfully.</i>
JSON	<i>201: { "status": true } - successfully registered the service 200: { "status": false } - either the service key is already used or an error occurred</i>

Table 4-2: Send Task Request Endpoint

<i>Send Task Requests</i>	
Description	<i>This endpoint enables users to send a list of task requests to a Resource Application Scheduler.</i>
<i>Request</i>	
URL	<i>POST http://fcn1:8080/reasoner/taskRequests</i>
JSON	<i>{ "duration": 5, [{ "serviceType": "t1", "serviceKey": "busy-image", "cloudTask": false, "fogTask": false }, { "serviceType": "t2", "serviceKey": "temp-hum", "cloudTask": false, "fogTask": true }] }</i>
<i>Response</i>	
Description	<i>Returns a status message that indicates whether the services, needed to execute the task requests, could be deployed.</i>
JSON	<i>201: { "status": true } - successfully deployed the needed services 200: { "status": false } - either an error occurred, or the fog colony is overloaded and no cloud-fog middleware is connected</i>

Table 4-3: Get Resource Utilization Endpoint

<i>Get Resource Utilization</i>	
<i>Description</i>	<i>Enables the Resource Allocation Scheduler to get the resource utilization of a specified device.</i>
<i>Request</i>	
<i>URL</i>	<i>GET http://fc1:8081/localdb/utilization</i>
<i>Response</i>	
<i>JSON</i>	<i>200: { "cpu": 0, "ram": 0, "storage": 0 } - not yet monitored device 200: { "cpu": 12.34, "ram": 45.23, "storage": 10.87 } - monitored utilization</i>

Table 4-4: Resource Allocation Scheduler Propagator Endpoint

<i>Propagate Service Data and Task Request</i>	
<i>Description</i>	<i>This endpoint enables Resource Allocation Scheduler to propagate task requests and general data to fog nodes if the task request is time sensitive. In case of task request not time sensitive this endpoint is used to send to the cloud-fog middleware.</i>
<i>Request</i>	
<i>URL</i>	<i>POST http://fc1:8080/propagator/propagate</i>
<i>JSON</i>	<i>{ "sender": "<device : FogDevice>", "key": "<key : String>", "data": "<list : List<HashMap> >" "requests": "<requests : List<TaskRequests> >, ..." }</i>
<i>Response</i>	
<i>JSON</i>	<i>200: empty</i>

Table 4-5: Cloud Fog Middleware Propagator Endpoint

<i>Propagate Task Request to the Cloud</i>	
<i>Description</i>	<i>This endpoint enables Resource Allocation Scheduler to propagate task requests to the cloud-fog middleware.</i>
<i>Request</i>	
<i>URL</i>	<i>POST http://cfm:8082/propagator/propagateTaskRequests</i>
<i>JSON/</i>	<i>{ [{ "serviceType": "t1", "serviceKey": "busy-image", "cloudTask": false, "fogTask": false }, { "serviceType": "t2", "serviceKey": "cloud-service", "cloudTask": true, "fogTask": false }]}</i>
<i>Response</i>	
<i>JSON</i>	<i>200: empty</i>

4.3 Implementation

This section explains the practical implementation of the RAS in fog computing framework for the IoT environment. The execution of the framework is explained, and some sampled necessary commands are given. This also allows the execution part to be reproduced. Detailed information on how the developed fog computing framework for the IoT environment can be executed is given in the following section. This section plays a pivotal role in answering the objectives.

4.3.1 Testing of an IoT Service.

This section explains how the new service that was developed in the fog landscape was registered and executed. To do so, there is no need to have knowledge of the infrastructure, communication and service deployment. The developed RAS in fog computing framework for the IoT environment enables users to develop services by providing:

- **A service key** whose function is to identify a Docker image uniquely
- **Docker volumes** whose function is to allow the container to utilise the resources that are hosted in the host file system.
- **Ports** whose function is to expose and flag all containers that require rights which help the Docker Container to have the same capabilities with the host.
- **Docker file**

As soon as the Docker image information has been elicited, the JSON file registers the service. The JSON file formed demonstrated below is responsible for registering service keys that will pull specific repositories and executes the *service.ny* file.

```
firs-app 1 {
        .pio 2 "serviceKey": "temp-hum",
        include 3 "dockerfile": "FROM jonasbonno/rpi-grovepi\n
        REA 4 RUN pip install requests\n
        lib 5 RUN git clone https://github.com/keyban/fogservice.git\n
        REA 6 ENTRYPOINT [\"python\"]\n
        src 7 CMD [\"fogservice/service.py\"],
        test 8 "volumes": [\"/dev/i2c-1:/dev/i2c-1\",
        REA 9 "exposedPorts": [\"8105\"],
        10 "privileged":true
        11
```

Code 1: JSON File that Registers a Service

In the above JSON, which was a sensor that monitors temperature and humidity in a room, a volume mapping is listed, and 8105 is the port that was exposed. It should be noted that privilege rights are wanted by the service to work correctly. Registration of the Docker Image is required in the framework to enable execution and evaluation of the service. In a bid to do that, the user sends a POST HTTP request to the Resource Allocation Scheduler, which is the control point in the system. The user requires IP address information to the RAS running in the gateways landscape. Having the IP address and API URL, then the URL `http://<IP>:8080/shareddb/register` is built. Whenever the request is sent, the content type is added as a header, and the created JSON file will be the body as demonstrated below with the HTTP data-transferring tool.

```
firs-app 1 curl -H "Content-Type: application/json; charset=UTF-8" -X \
        .pio 2 POST -d @image.json http://<IP>:8080/shareddb/register
        include 3
```

Code 2: HTTP data transferring tool

When the URL is correct, and the RAS is up and functioning properly, then the response contains the URL and the status flag indicating if the service could be created or not. In the event the flag is false, then there is a possibility that either the service-key is assigned already or there is some other error that occurred. Therefore, they will need to check error information in the RAS. In the event that the registration was successful, the request execution of the registered service will be requested. A JSON application has to be created and send to the RAS to achieve the deployment of the registered service. The generated application comprises of total service duration, which is used to define when the service has to be stopped and contains a list of tasks request which identifies the services to be deployed. The interval field can be set to

infinity by -1, and this field is in minutes. A task request contains service type, service key and a flag that indicate whether it is time-sensitive or not. If it is time-sensitive, it will be deployed in the fog layer, and if not time-sensitive, it will be deployed to the cloud. The URL `http://<IP>:8080/reasoner/taskRequests` is used to send this request. Below are the examples of the command and corresponding JSON file, which is required in the execution of the request.

```
1 curl -H "Content-Type: application/json; charset=UTF-8" -X \
2 POST -d @app.json http://<IP>:8080/reasoner/taskRequests
3
```

Code 3: Command used to Send the Task Requests for Execution

To request a task, the following has to be used.

```
1 {
2   "duration": "-1",
3   "requests": [
4     {"serviceType": "t1", "serviceKey": "temp-hum",
5      "cloudTask": false, "fogTask": true}
6   ]
7 }
8
```

Code 4: Task Request Application JSON

After deployment, there will be a JSON response stating that the service was successfully deployed and will be comprised of two things that are i) a URL header and ii) a payload marked with deployment time in milliseconds (ms). This will help when you want to investigate either the running device or the status of the HTML webpage. The status web page contains device-dependent information including IP, port, device type, children, parent, registered Docker Images, and running Docker Containers. The URL "`http://<IP>:<PORT>/`" which is comprising of the corresponding device IP and device port is used to access the status web page of the device.

Important nodes used in the Node-RED are the function node, inject node, (Message Queuing Telemetry Transport (MQTT) node, switch node and the debug window which shows the status. Below is an example of Node-RED flowchart.

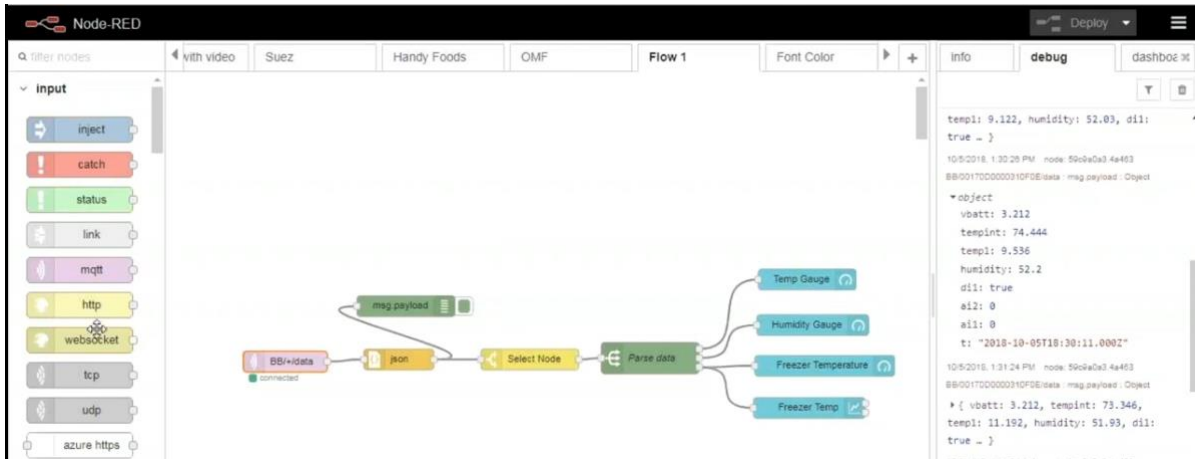


Figure 4-9: Node-RED flowchart

In the home assistant environment, the event state, current state and call service were used. Another example was for the automation of a sensor that will send a message, picture and highlight which window was opened. New constrained can also be defined in Node-RED.

<p>Delete Cancel Done</p> <p>node properties</p> <p>Name</p> <p>Data</p> <p>Function</p> <pre> 1 msg.payload = 2 * { 3 * "data": { 4 * "message": msg.windowName, 5 * "data": { 6 * "push": { 7 * "badge": 5, 8 * "category": "siren" 9 * }, 10 * "attachment": { 11 * "url": msg.image, 12 * "content-type": "jpeg", 13 * "hide-thumbnail": false 14 * } 15 * } 16 * } 17 * } 18 return msg; </pre>	<p>Edit function node</p> <p>Delete Cancel Done</p> <p>node properties</p> <p>Name</p> <p>Data</p> <p>Function</p> <pre> 1 msg.payload = 2 * { 3 * "data": 4 * { 5 * "message": msg.payload 6 * } 7 * } 8 return msg; </pre>
--	--

Code 5: Open Window Notification

4.3.2 IoT Application Execution

If there are already registered Docker Images of an IoT application, it will be easier to deploy several services for execution. A JSON file for the application was built and sent to the same URL. To form an application, the JSON file should fulfil the following task request.

```

1  {
2  ...
3  "duration": "5",
4  "requests": [
5  {"serviceType":"t1", "serviceKey":"temp-hum",
6  "cloudTask":false, "fogTask":true},
7  {"serviceType":"t1", "serviceKey":"temp-hum",
8  "cloudTask":false, "fogTask":true},
9  {"serviceType":"t2", "serviceKey":"busy-image",
10 "cloudTask":false, "fogTask":false},
11 {"serviceType":"t4", "serviceKey":"cloud-service",
12 "cloudTask":true, "fogTask":false},
13 ]
}

```

Code 6: JSON file to form an application.

For an application to be deemed successful, all the task requests have to be deployed without error. In the event that there was an error in one task request, then the whole application will not be deployed, resulting in stopping the already deployed services. To check for results of deployment, you request the cloud database by typing URL: “<http://<IP>:8200/db/>” or check on the debug window on the Node-RED home page.

4.3.3 Testing of the Resource Allocation Scheduler

The Resource Allocation Scheduler is at the centre of this research. Resource Allocation Scheduler role is to make a decision as to where task request is to be assigned either to the fog layer or to the cloud layer depending on the status. In previous solutions, the IoT devices would send the task to a fog node and the fog nodes will make a decision to either process the whole task, part of the task or offload to another fog node or cloud which compromised QoS in fog computing. To modify and improve the algorithms used in previous approaches to suite Resource Allocation Scheduler, a java class was created which provides the following interface in the newly introduced RAS.

```

1  public interface IResourceAllocationScheduler {
2  ApplicationAssignment handleTaskRequests (
3  Set<IoTdevice> chi ldr en , Set<TaskRequest> r e q u e s t s )
4  throws InterruptedException , Exception ;
5  }

```

Code 7: Resource Provisioning Java Interface

4.3.4 Evaluation Round-Trip Time

After making sure that everything was up and running, 1000 independent runs were done; results collected and averaged for each parameter which was tested. In particular, we were concerned about determining the round-trip time. To determine round-trip time, this study focused on queueing time and offloading time. It is important to reiterate that two important codes play a pivotal role in recording the start time at each node and calculate the time elapsed in each sequence (iteration).

```
1 context.global.startTime=new Date().getTime();
2 return msg;
```

Code 8: Code that saves the start time of the flow at each node.

The time elapsed message is generated by the following code. This code considers the message that would have passed through a node that saved the start time function.

```
1 var currentTime=new Date().getTime();
2 var timeElapsed=(currentTime-context.global.startTime)/1000;
3 msg.payload="Time elapsed is:"+timeElapsed+"s";
4 return msg;
```

Code 9: Code that calculates the elapsed time.

The results always appear on the debug window in the Node-RED as shown below.



Code 10: Debug Window

It is the benchmarks in Node-RED that report the time taken to run all matching in/out action flows for one given iteration. Many debug outputs can be generated and displayed at the debug window, stored and presented in the dashboard.

4.3.4.1 Queueing Time

In order to assess queueing time which is the time a task waits in the queue before it is assigned to a fog or cloud resource based on its priority category (high, low and no priority tasks); tasks were sent automatically and periodically from IoT devices at the same time in batches of 5 after every 10 ms time-stamp. A maximum of 40 tasks was assigned for each of the three categories of the tasks.

Based on the set-up parameters, when a task is automatically deployed from an IoT node to the RAS, the task is time-stamped with the time it leaves the IoT node (t_1) which is the start time and the time it arrives in RAS (t_2). When that task is deployed from the RAS to either fog node or cloud, it is also time-stamped with the time it arrives at either fog or cloud (t_3). All these times will be recorded and appear at the debug window in Node-RED for each task respectively. Therefore, the queueing time (Q_t) will be calculated and estimated automatically using the formula $Q_t = (t_2 - t_1) + (t_3 - t_2)$ which is the elapsed time.

The results for each task category were recorded in milliseconds, collected and displayed at the debug window. At first, there was variability in the measured queueing time of the same task, but after running the experiment for 1000 runs, a correlation is noticed. After queueing time has been automatically calculated, the averaged queueing times for each task are stored in files and shown in the dashboard. It is from these recorded times where graphs in the results section were plotted.

4.3.4.2 Offloading Time Evaluation

To evaluate overall offloading time, which is defined as the time taken to upload a task (equal to the queueing time), process a task and download a task response at the IoT device; tasks were sent automatically and periodically from IoT devices. They were sent at the same time in batches of 5 tasks after every 10ms time-stamp to the RAS then fog or cloud and back to the IoT device. A maximum of 40 tasks was sent for each of the three categories of the task.

In this case, the uploading time is equal to the queueing time (Q_t). Process time (P_t) is time we get from the initial time the task arrives at the fog or cloud (stamped as t_3) to the time the task or response leave the resource stamped as t_4 (the elapsed time). Therefore, the processing time is $P_t = t_4 - t_3$. Downloading time (D_t) is the time the tasks

leaves the processing resource to the time it arrives at the IoT device and is stamped as t_5 . Therefore, downloading time is $Dt = t_5 - t_4$. Offloading time (OT) can be calculated as $OT = Qt + Pt + Dt$ which is the overall elapsed time. All these times t_1 to t_5 will be appearing at the debug window in Node-RED for each task respectively. The results for each task category were shown at the dashboard and recorded in seconds. After offloading time has been automatically calculated and averaged, the queuing times for each task are stored in files and shown in the dashboard. It is from these recorded times where graphs were plotted to show the impact of RAS in overall offloading time.

Below is an example of some of the average times obtained for delay per user, offloading time and queing time which were used to come up with the graphs in chapter five.

Average delay per user(seconds)			
Packet Arrival Rate (packets/second)	Entirely processed at IoT device	Fixed offload to Fog node	RAS approach
0	0	0	0
5	0.3	0.4	0.35
10	0.6	0.7	0.65
15	0.92	0.92	0.92
20	1.26	1.18	1.17
25	1.58	1.42	1.34
30	1.88	1.65	1.44
35	2.2	1.68	1.42

Offloading Time (seconds)			
Number of Task	High Priority tasks	Low priority task	No priority
0	0	0	0
5	1.19	1.25	1.39
10	1.25	1.37	1.7
15	1.65	1.85	2.3
20	2.3	2.6	2.8
25	2.8	3.3	3.45
30	3.5	3.7	3.95
35	4.1	4.3	4.45
40	4.7	4.75	4.9

Average delay per user(milliseconds)			
Number of Tasks	High Priority Tasks (Hard deadline tasks)	Low Priority Tasks (Soft deadline tasks)	No Priority Task (No deadline task)
0	0	0	0
5	5	10	15
10	10	20	30
15	15	30	45
20	20	40	60
25	25	50	75
30	30	60	90
35	35	70	105
40	40	80	120

It is important to reiterate that the above times are averaged time after 1000 runs.

4.4 Conclusion

This chapter presented the framework design of this research, in particular, components of the framework and their functions, aspects expected to be supported by the framework and workflows used in this thesis. The implementation of the whole framework was presented emphasizing IoT services, application execution, testing RAS and evaluation of round-trip time (queuing and offloading time). In light of the evidence provided in this chapter, the chapter managed to prove and answer the research question “Can an adaptive resource allocation scheduler in fog computing framework which is based on tasks requirements and priorities be successfully developed?”

Chapter Five: Results and Discussions

5.1 Introduction

This chapter gives answers to this study's fifth research question "What is the performance of the adaptive RAS in fog computing framework?" The chapter begins by presenting an overview simulation configuration in **Section 5.2**. Followed by a detailed presentation of the study outcomes obtained through simulation tests in **Section 5.3**. In particular, we were looking at queuing time and offloading time which are factors that affect round-trip time. Finally, **Section** Error! Reference source not found. will present a critical discussion relating our findings to the theory and literature review. **Section 5.4** concludes the chapter by giving a synopsis of the whole chapter.

5.2 RAS Performance Evaluation Setup

This section presents the simulated environment description and assumptions.

5.2.1 Simulated Environment Description

The study's simulation was hosted on a high-performance computer with 1100 terabyte (TB) storage capacity, 135 cluster nodes with 2900 processor cores and 11TB memory. The simulation parameters of the framework are shown in **Table 5-1**. As highlighted in the table, 20 IoT devices were used for this simulation and generated multiple tasks at a specific time interval. The input data size was between 10MB to 30MB, and for output, data size was 1MB to 30MB. Both input and output data size was uniformly distributed. Twenty (20) Mbps was used as maximum transmission bandwidth from the IoT device via the RAS to either remote fog or to cloud servers in both data uploading or downloading. All these conditions and parameters were maintained at constant. For the simulation, 1000 independent runs were done and averaged for each parameter to get a better result output for the runs. **Table 5-1** shows the simulation parameters used in this research.

Table 5-1: Simulation Parameters

Parameters	Values
Number of IoT device	20
Number of IoT-Fog Gateways	3
Number of fog nodes	10
Number of cloud data centres	2
Number of tasks	100
IoT device CPU frequency	600x10 ⁶ cycles per second
IoT device memory capacity	128 Mega bytes
Fog nodes CPU frequency	5 x 10 ⁹ cycles per second
Fog nodes memory capacity	512 Mega bytes
Cloud server's CPU frequency	10 x 10 ⁹ cycles per second
Cloud server's memory capacity	64 Giga bytes
Maximum Bandwidth	20Mega Hertz

5.3 Performance Evaluation.

The study focused on addressing resource allocation challenge which affects latency, a factor of the round-trip time for packet delivery in the IoT environment. The experiment considered the queuing time, offloading time and throughput during performance evaluation. Another thing considered was the starvation problem which was being experienced by low priority tasks.

5.3.1 Results

The results focused mainly on roundtrip time parameter which is queing time and offloading time.

a) Queuing Time

As defined earlier, queueing time is the time a task waits in the queue before it is assigned to a fog or cloud resource. Queueing time plays a pivotal role in determining whether a task will be processed early or not based on how long a task would wait before it is assigned to a resource. The higher the queueing time, the higher the chances of an increased round-trip time, which has a negative impact on latency and affects QoS. High latency has a negative impact on time-sensitive tasks. Consequently, queueing time should be minimized to reduce round-trip time which in

turn reduces latency. In the study's experiment, we simulated the queuing time for the high priority, low priority and no priority tasks. **Figure 5-1** presents the result from that simulation.

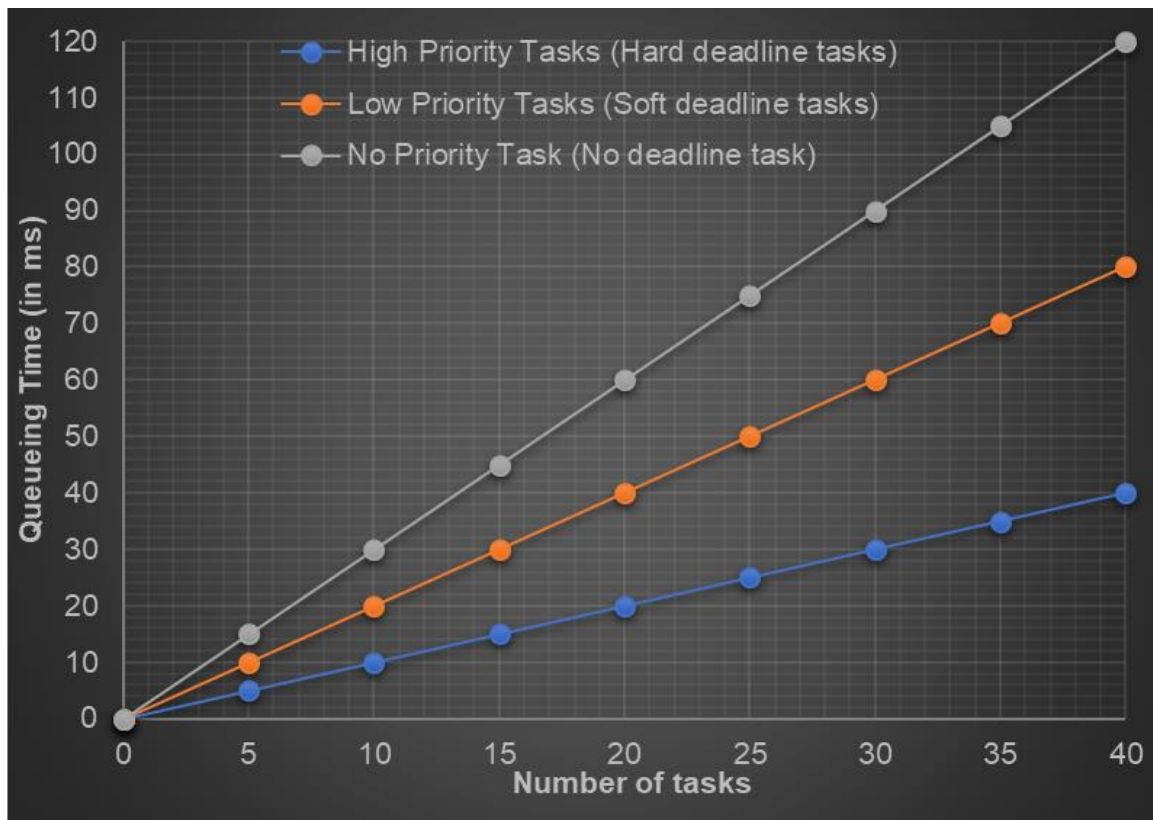


Figure 5-1: Queueing time for high priority tasks, low priority tasks, and no priority tasks

Based on **Figure 5-1**, it can be noted that for high priority-based tasks (blue line), the queuing times are minimal when compared to low priority-based tasks (orange line) and no priority-based tasks (grey line). Similarly, low priority-based queuing time is also minimal when compared to no priority-based tasks. This is because high priority tasks are given high preference by RAS during the assignment to both message routing and the fog layer resources to be processed first as compared to the later. For this reason, high priority tasks are assigned and processed earlier than the other two, which gives them less queuing time. Correspondingly, the low priority tasks are given a better priority compared to those with no priority.

Queueing time for the next higher priority task is affected by the time taken by the RAS to decide whereas for low priority task, queuing time is directly affected by the number of high priority tasks on the queue and the time taken by the RAS to make a decision.

The more the number of high priority tasks in the queue, the more the queuing time for low priority task. However, with the queueing model introduced at the RAS as explained in framework design chapter, the queueing time of low priority tasks was reduced due to the fact that after sometime stamp, if there are still high priority tasks on the queue, low priority task in the low priority task queue was promoted to the high priority queue. This was done to address the starvation problem that was being faced by low priority queues when there are more high priority tasks which keep coming from the IoT devices.

When considering no priority task, the queuing time is affected by the time it arrives at the RAS to be assigned to the cloud, the size and offloading speed of the first task in the queue to be offloaded to the cloud. As a result of this, the bigger the task to be offloaded to the cloud and the time it arrives at RAS, the greater the queuing time for those tasks that should be offloaded to the cloud.

b) Offloading Time

Offloading time is another factor that affects round-trip time. Offloading time is the time taken to upload, process and download a task from IoT device to RAS then either fog node or cloud depending on the task status. The more the offloading time, the greater the overall round-trip time. Moreover, offloading time is directly affected by queuing time. If the queuing time is minimized, the overall offloading time is also reduced.

Figure 5-2 presents the simulation result of offloading time.

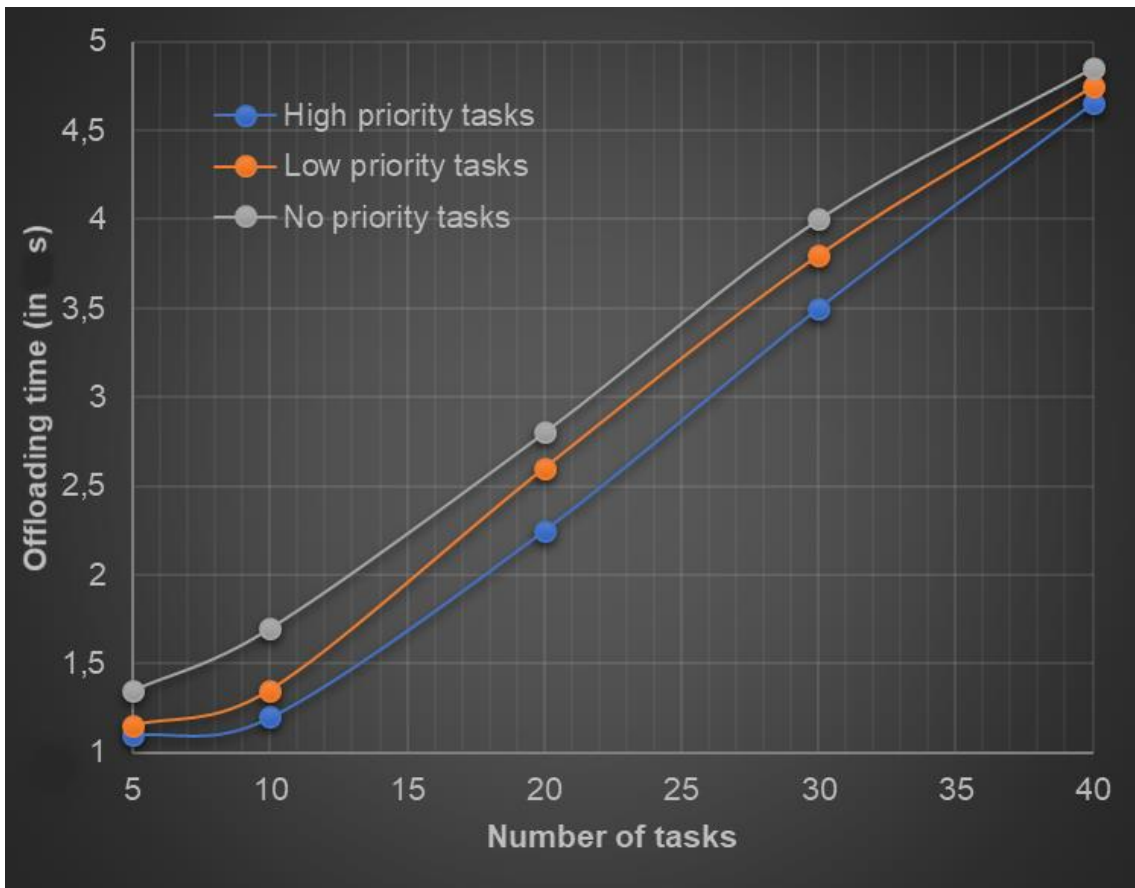


Figure 5-2: Offloading time for High priority tasks, Low priority tasks, and No priority tasks

As can be noted, **Figure 5-2** shows that the average offloading time of task to a resource increases as the number of tasks increases. From the graph, it is noted that offloading started happening after one second (1s). This was because some delays were experienced when tasks were sent from the IoT devices to the RAS. Another point to note is when high priority tasks and low priority tasks are less than 5, they can be processed in the IoT device itself, hence no need to offload them to the RAS.

As shown in **Figure 5-2**, for 10 tasks the offloading time for high priority tasks (blue line) and low priority tasks (orange line) is less than 1.5 seconds whilst no priority tasks (grey line) has over 1.5 seconds. As the tasks are increased with a factor of 10 to 20 tasks, high priority tasks have an offloading time of approximately 2.3 seconds whereas low priority tasks and no priority tasks are above 2.5 seconds but below 3 seconds. For 30 tasks, the offloading time for high priority tasks is approximately 3.5 seconds, whereas low priority tasks and no priority tasks are at approximately 3.8 seconds and 4 seconds, respectively. At 40 tasks, the offloading time of high priority tasks, low priority tasks and no priority tasks is at approximately 4.7 seconds, 4.8

seconds and 4.9 seconds respectively. From the graph, it can be noted that the increase of offloading time in seconds was directly proportional to the rise in several tasks by a factor of approximately 10 from 10 tasks to 40 tasks. This was not the case for low priority tasks and no priority tasks. In a nutshell, the offloading time of all the tasks would increase as the number of tasks increased. High priority tasks have lower offloading time when compared to the other two because they were given first preference to fog layer resources. Even though the no priority tasks were not being offloaded to the fog layer, they took more time to be offloaded to the cloud. The reason behind this is because of their size, which requires more time in offloading to the cloud. The bigger the task, the more the time it took to offload it.

c) RAS Strategy Versus Other Resource Allocation Strategies.

i) Comparison of processing in the IoT device, fixed offloading to fog node and the use of RAS strategy

For comparison, we compared the average delay per user against packet arrival rate using the fixed strategy method, RAS strategy and entirely processing the tasks on the IoT device. A fixed strategy is when tasks are sent directly to the fog nodes from IoT devices, and the fog nodes would decide to either process the whole task, part of it or to send to the cloud. This strategy is the one being used by many researchers in literature. **Figure 5-3** presents an illustration of task data arrival, task execution and task offloading.

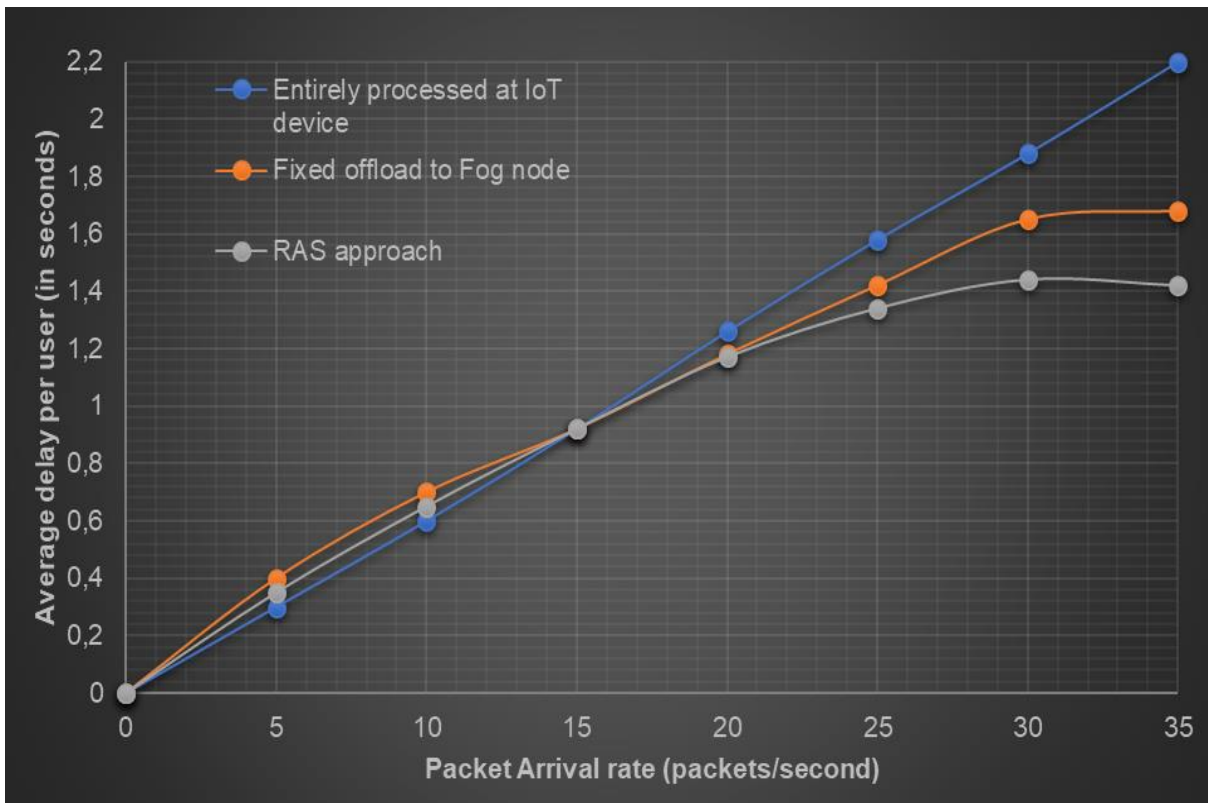


Figure 5-3: Illustration of task data arrival, task execution and task offloading

In **Figure 5-3**, it can be noted that if few tasks are time-sensitive, processing them in the IoT device is much better than transferring them to the fog node, as this increases the round-trip time. Another thing that **Figure 5-3** reveals is that, at a packet arrival rate of between 0 and 14, the average delay per user is less when the task is entirely processed in the IoT device itself than when it is sent to either the RAS or fixed offloading to the fog node. RAS strategy is better when compared to fixed offloading. This is because RAS can then assign a task directly to where it is supposed to be processed when compared to the later.

At 15 packets/sec arrival rate, the average delay per user is the same for all the three strategies. This convergence point can be referred to as a point of equilibrium. This is so because, at 15 packets arrival rate, the summation of the delays and the resources available at both the fixed offload strategy and the RAS strategy would be equal to the one processed entirely at the IoT device. Since IoT devices have limited computational power, the more the tasks produced, the more the time needed to process those tasks. This is also the case with the fixed offload. The delays increase if decisions are made in the fog node. This point is referred to as the saturation point where the tasks produced is equal to the resources. In this case, any other tasks being deployed has

to wait on the queue.

After 15 packets/sec arrival rate, using other resources that is fog node resources will be of greater benefit as can be noted in the diagram. After 15 packets/sec arrival rate, the average delay per user of those tasks that are processed entirely in the IoT, keep on increasing when compared to the other two options. This is because IoT devices would no longer have the capability to process the tasks, as some of the tasks on the queue in IoT devices might be CPU intensive. An increase in task production and packet arrival rate has a greater effect when the task is processed entirely at the IoT device level. The more the packets in the queue to be processed, the more the time needed to process them, especially when they are processed in the IoT device itself. The packet arrival rate is directly proportional to the average delay per user if the task is processed in the IoT device. The demand for computing resources by the tasks can even cause the IoT device to end up being slow and not working properly. This can also have a negative impact on the IoT device battery lifespan, as the device will be strained, which results in the device using more battery power.

It is also worth mentioning that when the tasks are sent from the IoT device to be processed on the fog node, which is randomly chosen, at first the round-trip time is increased even though there would be a lower processing time in the fog layer. The round-trip time is increased because some delays are encountered during transmission time as a result of the offloading part from the IoT to the fog node. In some cases, the task that is sent to the fog node e.g. **fog node A**, will not meet the processing requirement of that node, as such it has to be offloaded entirely to the other **fog node B**, or part of the task is processed in **fog node A** and the other offloaded to **fog node B**. Although at a low packet arrival rate, the average delay in the fixed offload strategy is higher than the processing at the IoT device, the value remains below 1s. This is because as soon as the packets arrive, they will be processed quickly since fog nodes have more processing power as compared to the IoT device. When the packet arrival rate increases, the fixed offload reduces the average delay by offloading to the fog nodes. After 15 packets per second, the fixed offload offers some advantages compared to those that are entirely processed at the IoT device.

Again, referring to **Figure 5-3**, it can be noted that between 15 to 20 packets per second arrival rate the performance of the fixed offloading and the RAS strategy are

almost the same. The impact can only be noted after 20 packets per second arrival rate when the RAS becomes better than the fixed offloading. The reason might be that the RAS chooses the best fog node for time-critical tasks when compared to the fixed offloading strategy. When tasks are using the RAS, it can be noticed that at first, using the RAS strategy will only be better if compared to fixed offloading but worst when compared to those that are processed at the IoT devices. The delay experienced in time is because some time is used when transferring a task from the IoT device to the RAS before the actual processing of the tasks starts. The difference is noticed when there is an increase in the packet arrival rate where it can be observed that the RAS outclass both strategies highlighted earlier. The reason being that the RAS chooses appropriate fog node to process the tasks when compared to the fixed offload strategy, which sometimes offloads tasks to a fog node that does not satisfy the requirements of the tasks which leads to some delays. As such, choosing the correct fog node that suits the requirement of the task first before assigning those tasks helps in the sense that when tasks are then finally deployed, it is guaranteed that they will be processed.

In light of the above findings, the study discovered that when time-sensitive task are less than 15 packets, it is of benefit to process them in the IoT device. When tasks are more than 15 packets and are both time-sensitive and non-time-sensitive tasks, it is better to forward to RAS for resource allocation part. By so doing, round-trip time would be reduced significantly.

ii) RAS strategy compared to other strategies

For comparison purposes, three strategies which are proposed and used in literature are considered and compared with RAS:

- **Strategy 1:** IoT devices would randomly choose a computing device either in the fog layer or cloud layer. Let us denote this scenario as **S1**.
- **Strategy 2:** IoT devices would choose a computing device with minimum uploading time. Let us denote this scenario as **S2**.
- **Strategy 3:** IoT devices would choose a computing device with sufficient CPU frequency for processing the tasks. Let us denote this scenario as **S3**.
- **Strategy 4:** Using the proposed Resource Allocation Scheduler strategy denoted with RAS.

- *Performance-based on average queueing time*

In this comparison, the tasks that are processed locally in the IoT devices are excluded. In most instances, IoT devices are known for producing time-sensitive tasks. It should be noted that RAS immediately schedules tasks with deadlines for offloading in the fog layer whilst no priority tasks are offloaded to the cloud. In most instances, IoT devices are known for producing high priority tasks or low priority tasks. Because of that reason, RAS was introduced, and simulations were done to check if it will minimize overall queueing time and offloading time for each task which leads to the minimization of the overall round-trip time. **Figure 5-4** presents a performance based on average queueing time.

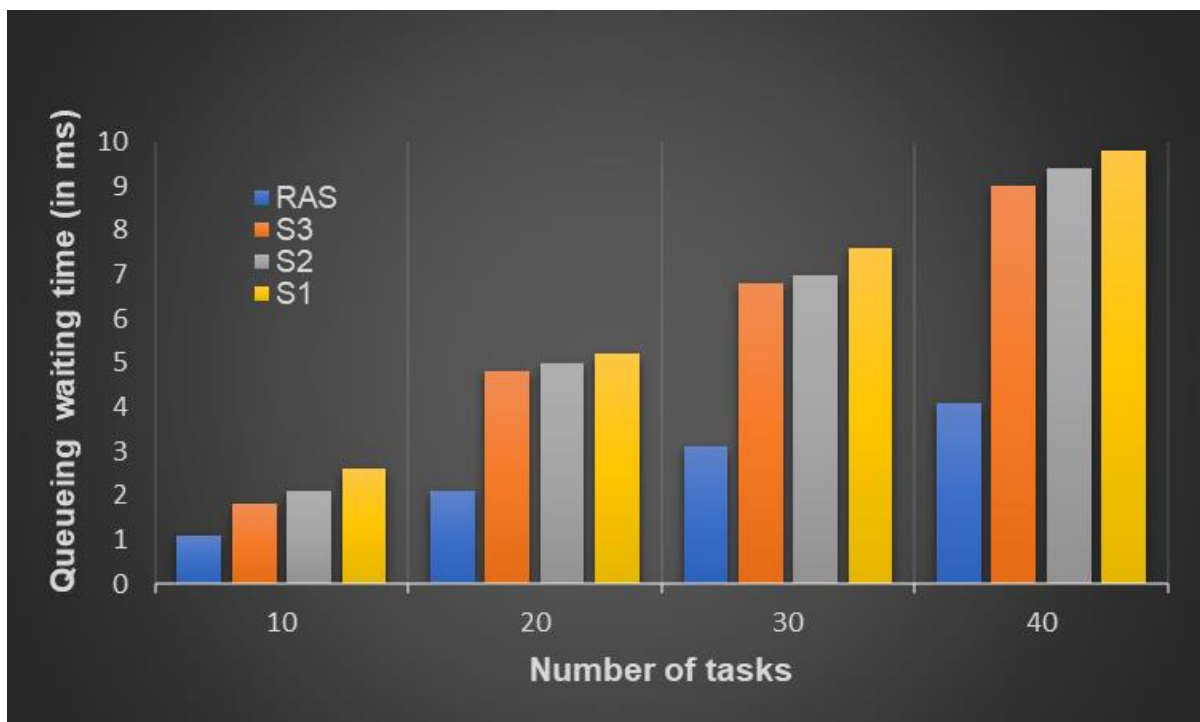


Figure 5-4: *Performance-based on average queueing time*

Based on **Figure 5-4**, which compares performance based on queueing time, our RAS improved performance when compared to the other three strategies as far as average queueing time is concerned. The average queueing time of high priority and low priority was minimized. This was as a result of them being given more priority if compared to those tasks with no deadline. It can be noted from **Figure 5-4** that even in the case of more tasks, the average queueing time of the RAS is less when compared to the other three strategies. These results proved that even if you are using first come first serve basis in different strategies if high priority tasks are not given high priority, that will

affect the queuing time and has a negative impact on the time-sensitive tasks as QoS is compromised.

- *Performance-based on average offloading time*

Even if **S2** strategy allowed IoT devices to choose a computing device with minimum uploading time and **S3** allowed IoT devices to select a computing device with sufficient CPU frequency for processing the tasks, it could be observed from **Figure 5-5** that these strategies did not minimize offloading time as expected by IoT devices. Contrary to **S1**, **S2**, and **S3**, considering performance based on the average offloading time as shown in **Figure 5-5**, RAS managed to deploy tasks to computing devices that met the requirements of the task. Moreover, RAS offered a minimum communication overhead, which minimized round-trip time since offloading time was reduced when compared to other **S1**, **S2**, and **S3** strategies. This was attributed to the fact that RAS would choose either fog node or cloud that satisfies the requirements of the task based on the task's status. Basing our argument on the simulation results, if the round-trip time is minimized, latency will also be reduced, and this will lead to improved QoS and improved performance.

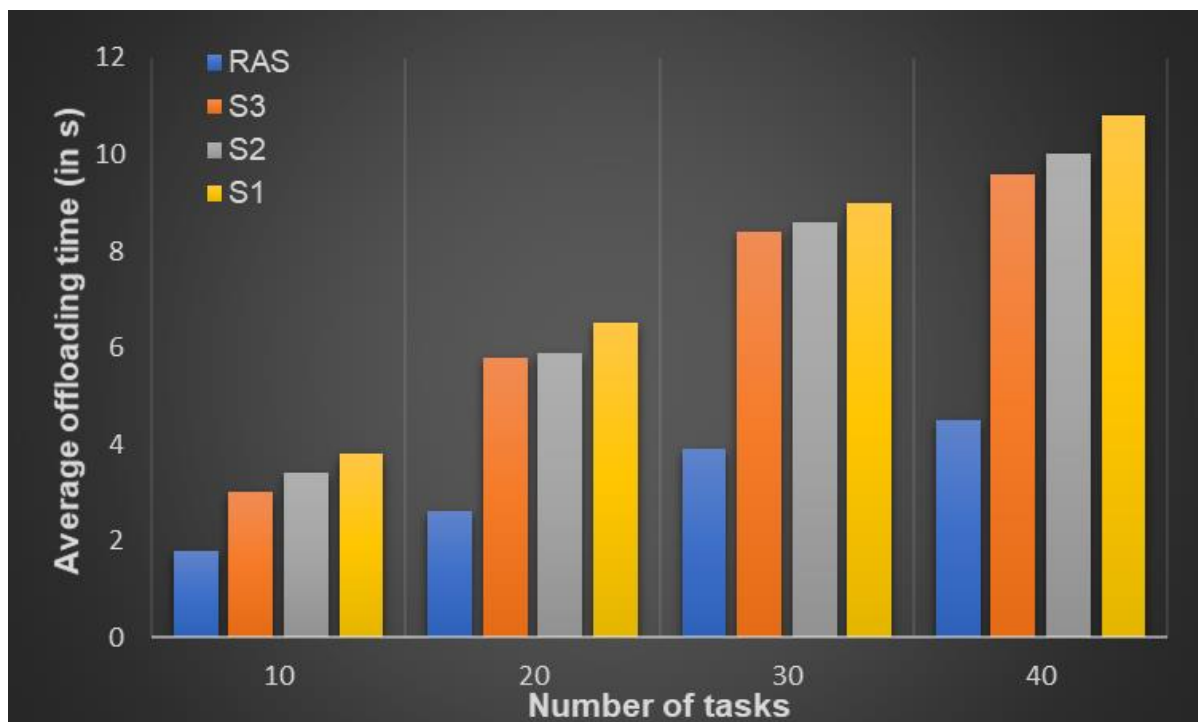


Figure 5-5: Performance-based on average offloading time

- Performance-based on average percentage number of tasks satisfying delay deadlines.

From the analysis given above in **Figure 5-4** and **Figure 5-5**, it can be noted that the number of tasks that meet their deadlines especially the high priority task and low priority task improved when using RAS compared to the other three strategies. When considering four scenarios, based on the number of tasks at a given time, it can be noted in **Figure 5-6** that when the tasks are few, and the RAS is used, high priority task can meet their deadlines by 100 percent. Whereas with the same number of tasks, other strategies' percentage is less than 90. **Figure 5-6** shows that RAS increased the number of tasks meeting the deadline by allocating high priority tasks to the fog resources followed by low priority. No priority tasks were assigned to cloud resources. This helped in improving network performance, as unnecessary congestion is reduced. Improvement of network performance has a direct positive impact on how tasks also transverses over the network. If tasks traverse easily in the networks, the round-trip time is also reduced.

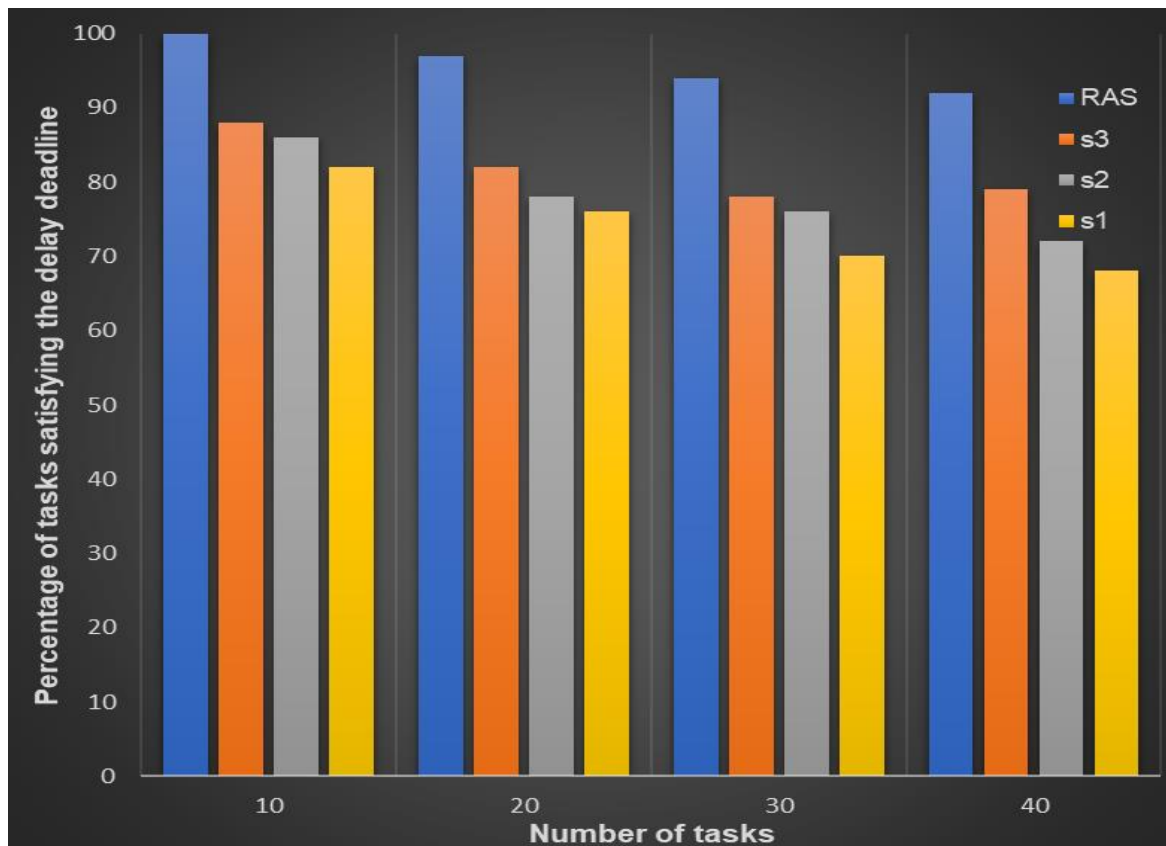


Figure 5-6: Performance-based on average percentage number of tasks satisfying delay deadlines.

- *Performance-based on average throughput*

In order to check if the RAS improved the QoS, throughput which is one of the QoS parameters was tested. In this work, throughput was calculated as the number of tasks that complete their process within a time-stamp based on the arrival rate. As indicated in **Figure 5-7**, RAS had high throughput when compared to other strategies. This is because different strategies failed to process more tasks within a given time-stamp. The RAS strategy managed to achieve improved throughput because it was able to deploy time-sensitive tasks to fog devices that met the resource requirements with minimum offloading time, which was also a factor of queueing time.

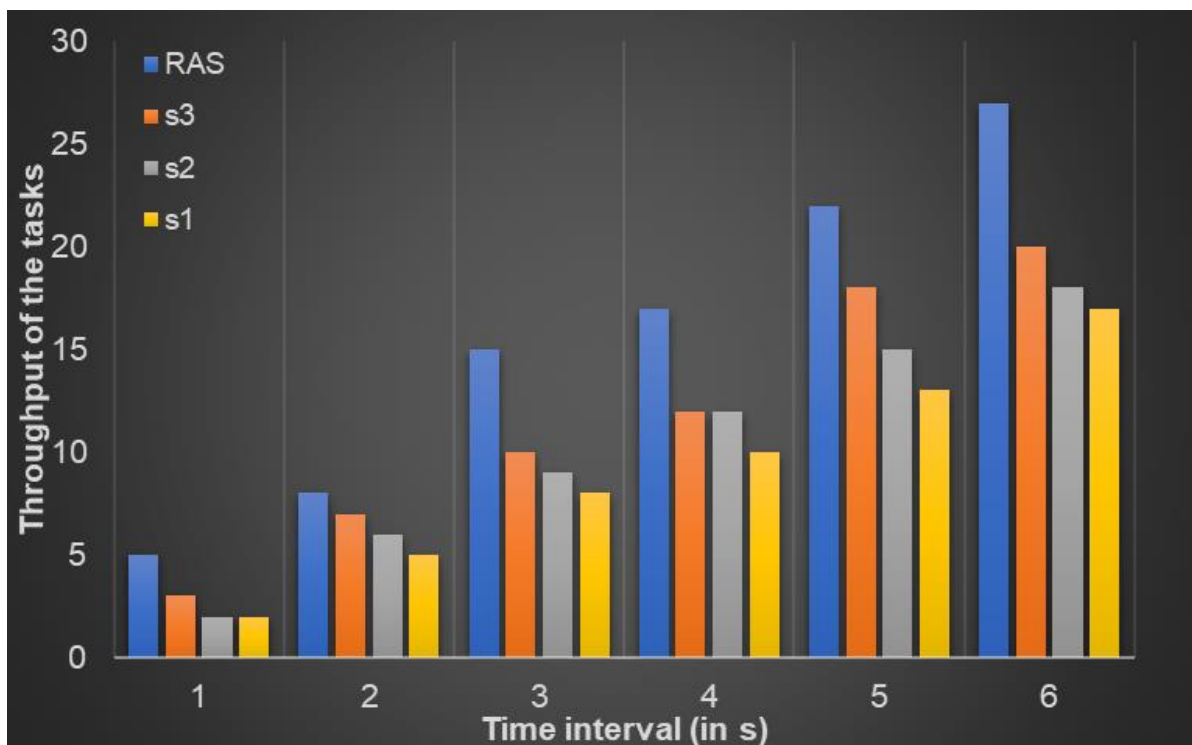


Figure 5-7: *Performance-based on average throughput*

5.3.2 Findings

The results above show that even though many factors play a pivotal role in determining the total round-trip time, queueing time and offloading time are very important too. Minimizing the two would help in reducing round-trip time which would result in the minimization of latency. In addition, if queueing time and offloading times are decreased, the overall throughput of the framework is significantly improved.

Another important finding from the study was that; choosing a computing device with sufficient CPU frequency for processing the tasks without considering other factors

such as the type of the tasks; would not minimize average queuing time and average offloading time in the case that more tasks are added. This kind of strategy would work well when there are few tasks that need to be processed. However, it might suffer when more tasks need to be processed.

Another point to note is that when choosing a device for processing, it is not wise to only consider the one with the minimum uploading time and ignore other factors like CPU frequency. As noted from the results, the strategy which only considers uploading time suffers a lot in minimizing queuing time and offloading time which are major factors in determining the reduction of total round-trip time.

CPU frequency of a device, uploading time of a device and the type of tasks to be uploaded in any device plays a critical role as far as queuing time and offloading time is concerned. It is also important to note that making a decision of which tasks is to be assigned to which resources; where the tasks are to be processed, at what time the task is to be uploaded and processed depending on their status whether they are time-sensitive or not; is very crucial in any framework.

From the results above, RAS was able to minimize queuing time and offloading time. Moreover, deadlines were met for both time-sensitive tasks and non-time-sensitive tasks. A starvation problem encountered by the low priority tasks was significantly reduced, and the throughput of both tasks was increased.

It is important to note that the queuing time and offloading time are not only the factors that affect round-trip time, but there exist some factors like device failure, connectivity, among other factors. Based on our findings while holding other factors at constant, RAS proved that it could help in reducing both queuing time and offloading time which are parameters that affect overall round-trip time.

- *Throughput*

As explained in **Chapter Two**, throughput is one such aspect that plays a pivotal role in the attainment of good QoS. Improved throughput leads to improved QoS. Throughput hinges on how many tasks will be processed entirely within a given time-stamp. It can be noted from the results above that high priority tasks were able to complete their operation and meet deadlines. However, that was not always the case for low priority tasks and no priority tasks. This was ascribed to the fact that the RAS

would deploy first the tasks that fall under the high priority category to be processed in the fog nodes with minimum communication overhead, which in turn improved the throughput of the high priority tasks. These results were a testimony to the fact that the introduced RAS improved the efficiency of the framework as far as the throughput of the time-sensitive tasks is concerned. Since RAS deployed high priority and low priority tasks in the fog nodes with minimum queueing and offloading time, this resulted in the minimization of the total round-trip time of time-sensitive task and increased the throughput.

5.3.3 Result Findings Analysis

Our results proved that when resource allocation is done properly, round-trip time can be minimized, and QoS in particular throughput can be improved as shown and explained above. The research managed to bring a positive contribution to what should be done to minimize queueing time and offloading time in an IoT-Fog-Cloud setup.

i) Queueing Time

It is evident from our findings that the queueing time for high priority tasks followed by the low priority tasks was significantly reduced when compared to other strategies suggested in the literature. This is attributed to the fact that RAS was able to decide first where (fog layer or cloud layer) and which resources to assign the tasks based on their priority, deadlines and resource needs. The assignment of resources based on priority helped in reducing network traffic which led to quick communication and reduced latency. In light of the above evidence from the results, RAS strategy in fog computing framework helped in minimizing queueing time.

It is also important to note that queueing time for both high priority tasks and low priority tasks can be further minimized when more fog nodes join the fog layer. The more the fog nodes at the fog layer, the less the queueing time to be experienced for high priority tasks and low priority tasks. The reason being that there will be more options and resources to assign the tasks. Therefore, queueing time for higher priority tasks and low priority tasks can be deduced to be directly affected by the number of fog nodes available at the fog layer. Apart from the factors highlighted above, undoubtedly, other factors might affect the queueing time as explained in the literature. These include the connectivity of devices, fog node failure, the CPU processing power of both the fog

nodes and the IoT-Fog gateways where the RAS is hosted among other software and or hardware challenges.

ii) Offloading time

Since offloading time is the time taken to upload, process and download a task from IoT device to RAS then either to fog node or cloud depending on the task status, minimizing queuing time plays a pivotal role in the minimization of overall offloading time. Since RAS strategy assigns tasks to a suitable resource in terms of processing power, it means the tasks will be processed faster, which will result in reduced processing time. It is a fact that reducing processing time will have a positive impact on the minimization of offloading time. Additionally, the fact that RAS gives high priority to network resources for time-sensitive tasks helps in reducing upload and download time of tasks which are all factors of offloading time.

As evidenced by the findings, offloading time for high priority tasks followed by low priority task was significantly reduced when using RAS compared with other strategies. This was because RAS would save time and fog node resources which were being wasted in deciding whether they can process the whole task, part of it or completely offload it to cloud as was the case in other strategies proposed in literature.

Grounding our reasoning from the findings and explanations above, it can be deduced that queuing time and offloading time plays a pivotal role in determining whether a task will meet its deadline or not. It was observed that high priority tasks always met their deadlines. Contrary, it is not always the case for low priority tasks as some tasks in this category missed their deadlines with few milliseconds. The delay is attributed to the fact that high priority tasks were given higher priority in the limited resources available in the fog layer as compared to low priority tasks. However, the impact of failing to meet the deadline for low priority tasks is not of greater importance, especially if the period is less than ten (10) milliseconds which is insignificant. It can be noted that in most cases, the requirements of most of the tasks generated by IoT devices, which are time-sensitive, are always satisfied because of the introduction of RAS in the gateways.

As a matter of fact, based on the results, it is true that introduction of RAS in the fog framework improved the throughput since throughput is the number of tasks that

complete their process within a time-stamp based on the arrival rate. Many of the tasks met their deadline, which shows it is a sign of improved throughput.

5.3.4 Discussion of Results Findings

In our simulation experiments, we reached a conclusion which was also supported by Souza *et al.*, (2017) who argued that QoS is not only affected by data transmission factors but also processing delays in fog nodes. We strongly agree with Souza *et al.*, (2017) findings that to reduce end-to-end delay in fog computing, fog nodes in the fog layer should be efficient and effective during processing. Otherwise, without doing that, even if you provide good data transmission factors on your network, an end-to-end delay which is overall round-trip time will still be high.

Our proposed RAS helped in reducing overall queuing time, offloading time and offered an improved throughput when compared to the strategies proposed by Wang *et al.*, (2019) and Mukherjee *et al.*, (2018). The two never considered the queuing time factor at the fog node, which also adds delay to the overall round-trip time. The present findings confirm the notion by Adhikari *et al.*, (2019) that queuing time and offloading time has an impact on the overall round-trip time. Therefore, this research agrees to the claims by Liu *et al.*, (2019), who stated that reducing queuing time and offloading time increases throughput and bring about improved QoS in fog computing.

Moreover, our results are a better true reflection of what overall round-trip time is because we also considered uploading and downloading time when measuring transmission delay, which plays a role in overall round-trip time. Contrariwise to the findings of Ko *et al.*, (2017), Tran *et al.*, (2017) and Mukherjee *et al.*, (2020) who ignored the downloading time and inter-fog transmission when they measured their transmission delay. Even though the abovementioned researchers' line of thinking of ignoring downloading time and inter-fog transmission was based on the reasoning that the delays are minor; our results proved otherwise that the slight delays when added, they contribute a lot in affecting round-trip time. In light of the above observation, we strongly argue based on our results that downloading time and inter-fog transmission should always be considered no matter how minor the delays may look like. Our findings conclude that those minor delays have a negative impact on the overall round-trip time, which in turn affect latency.

Furthermore, our simulation findings are in line and agreement with the findings of Alnoman and Anpalagan, (2018) who found out that prioritized queues minimize the delay for time-sensitive tasks as compared to non-prioritised queues. It can be observed from the results and conclusions of Alnoman and Anpalagan, (2018) that many low priority tasks were faced by the starvation problem. This was a result of low priority task missing their deadlines since only time-sensitive tasks were prioritised. To avoid the starvation problem which was faced in Alnoman and Anpalagan, (2018), we introduced a queuing model in RAS which would increase the priority of low priority task by one after a specific timestamp, thus promoting the low priority to be treated as a high priority task. As evidenced by our results, this played a critical role in addressing the starvation problem, which was experienced by low priority tasks.

Using RAS in the fog framework, we managed to process time-sensitive tasks and non-time sensitive at once without compromising time-sensitive deadlines. This was achieved with the IoT-Fog-Cloud architecture, which provided platforms suitable for the processing of CPU intensive tasks in the cloud and non-CPU intensive tasks in fog layer at once. This discovery goes in line with the ascertains of Chekired and Khoukhi, (2018). Chekired and Khoukhi, (2018) noted and concluded that using IoT-Fog-Cloud hierarchical structure will help in promoting a delay-tolerant network, which supports IoT data processing while meeting various QoS objective of both time-sensitive and non-time-sensitive requirements.

From the research findings, we can conclude that providing an excellent resource allocation; a method for QoS-aware fog networking and message routing plays a significant role as far as reducing round-trip time and improving throughput is concerned. Important points to note from our findings are: i) A proper resource allocation strategy in fog computing framework like RAS will help in minimizing round-trip time; ii) reducing queuing time and offloading time helps in reducing overall round-trip time which in turn reduces latency; iii) reducing queuing time and offloading time improves overall throughput and lastly iv) an adequately designed queuing model can help high priority tasks and low priority tasks to meet deadlines. Moreover, the starvation problem of low priority tasks will be minimised.

5.4 Conclusion

This chapter evaluated the benefits of using RAS strategy in fog computing framework, microscopically looking at queuing time and offloading time which are factors of round-trip time. The results show that introducing RAS in the fog computing framework helps in minimizing queuing time, offloading time which led to improved throughput. Furthermore, the queuing model introduced in the RAS helped to address the starvation problem, which was faced by low priority task. All these accrued benefits clearly proved that if resource provisioning is done well in fog computing, round-trip time can be minimized and QoS in particular throughput will be improved. This chapter managed to provide answers to the fifth research question “What is the performance of the adaptive RAS in fog computing framework?”

Chapter Six: Conclusion and Future Work

6.1 Introduction

This chapter reflects by articulating a review synthesis of the whole research's key points and how the research problem, objectives and research questions were answered. It further outlines the significance and shortcomings of the research findings. Scientific contributions brought by the research are emphasized. In conclusion, avenues for future research are suggested.

The chapter begins with a summary of the whole thesis by outlining the major points of each chapter in **Section 6.2**. In **Section 6.3**, a recap of research questions, research objectives and how they were answered is presented. The contributions of this research, the findings and their impact in the IoT-Fog-Cloud scientific space are presented in **Section 6.4**. The chapter is concluded with the future work in **Section 6.5**, which look at what should be done to improve our findings and future recommendations.

6.2 Summary of the Study

Without any doubt, as supported in literature **Section 2.4**, fog computing has played a pivotal role in addressing many cloud computing challenges in supporting IoT technology. However, the ever increase of IoT devices is a cause of concern to the fog computing technology. As such, many methods, approaches and strategies need to be designed and implemented in fog computing itself; if we want to continue harnessing from its advantages of offering minimized round-trip time, improved QoS and many other fog computing advantages.

In the current IoT-Fog-Cloud architecture, the tasks of different types are generated by IoT devices. These tasks are processed either in IoT devices themselves or offloaded to other computing devices in the fog or cloud layer depending on where the resources are available. These tasks are of different computational and QoS requirement. Thus, there is an apparent demand for novel resource allocation and offloading strategy that aim at minimizing the round-trip time under varying network traffic without compromising the QoS and expectations from the end-user.

In this regard, this thesis proposed and introduced a resource allocation scheduler in fog computing framework for the IoT environment. Below is how the study was done to end up having this thesis.

In **Chapter One: Introduction**, the research niche was established, a synopsis description of research aim and what motivated the study was highlighted. In order to address the research aim, research questions and research objectives were formulated which guided this study. The chapter was concluded by giving intended contributions and limitations of the study.

Chapter Two: Background and Related Work gave contextual knowledge, which acted as the foundation of the research. The chapter gave an insight into the IoT technology, cloud computing, fog computing and a clear view of how these three are intertwined. A point to be taken from the technologies is that cloud computing provides computational and storage power to IoT devices which are resource-constrained; whereas fog computing complements the cloud by bringing the cloud resources closer to the “ground” where the IoT devices are. Consequently, fog computing brings and offers security, cognition, agility, low latency, and efficiency [3]. Microscopically, a critical analysis of work done specifically in relation to QoS in fog computing was presented. This helped in identifying open challenges concerning QoS in fog computing which are research gaps. More precisely, the identified gaps represent the opportunities for further research in specific areas of fog computing systems and structures to enable IoT system executions. These research gaps acted as the basis of why there was a need to do this research and its importance in the scientific space. The open research gap, which was addressed in this chapter, was then highlighted. This chapter helped in identifying the functional, technical and non-functional requirements of the general IoT-Fog-Cloud architecture from existing works that were implemented. This chapter helped in answering the following research questions:

1. What are the key challenges in communication and computer resources allocation in an IoT environment?
2. How are communication and computing resources allocated and assigned among the IoT devices based on tasks, requirements and priorities?

Within **Chapter Three: Research Design and Methodology**, the procedures and techniques that were used for this whole thesis were presented. The chapter indicated

that the top-down research methodology was adopted for this work. The chapter also highlighted what was done at each of the four phases of the top-down methodology. This chapter answered the third research question of this study:

3. Which approaches can be used to build an adaptive resource allocation scheduler framework for IoT environment?

Chapter Four: Framework Design and Implementation presented the high-level design of the whole fog computing framework with more emphasis given to the Resource Allocation Scheduler (RAS), which was our contribution. The functional, technical and non-functional components of the whole framework and in particular the RAS were presented. Furthermore, the implementation of the framework which details the feasibility of the proposed framework was presented. This chapter gave a synopsis of how the service was deployed, the environment setup and the development, testing and execution of the framework. The chapter helped in answering the fourth research question:

4. Can an adaptive resource allocation scheduler in fog computing framework which is based on tasks requirements and priorities be successfully developed?

Chapter Five: Results and Discussions presented the evaluation of the implemented framework. This was done to illustrate and verify if the introduction of resource allocation scheduler in fog computing framework was of any benefit in reducing round-trip time. A critical discussion relating to the findings back to the theory and literature review was also done. This chapter was the answer to this study's fifth research question:

5. What is the performance of the adaptive RAS in fog computing framework?

All the above chapters gave an understanding of how the data gathered from literature was used to design, develop, implement and evaluate the framework. Thus, precisely helping in answering the research questions, research objectives while fulfilling the aim and showing our contribution to the scientific board of knowledge.

6.3 Research Objectives, and Where Addressed

Research objectives underpin this study, and as such, they should be alluded to, and how they were answered according to the full scope of this study.

In this work, five specific objectives were which were derived from our research questions were set:

1. To identify the key challenges of data communication and computer resources allocation in an IoT environment.
2. To determine how data communication and computer resources are assigned and allocated based on tasks requirements and their priorities in IoT environments.
3. To identify a suitable methodology for building a resource allocation scheduler in fog computing framework for IoT environments.
4. To build a resource allocation scheduler framework in fog computing for IoT environments.
5. To test and evaluate the proposed RAS in fog computing framework.

As can be seen above, the five objectives are interconnected and depend on one another. The above research objectives were formed from the alluded research questions. Below is a summary of tables which pinpoint in which chapter the research questions were answered.

Table 6-1: Research Objectives and Where they were Addressed

Research Objective	Chapter it was Answered
1. To identify the key challenges of data communication and computer resources allocation in an IoT environment. 2. To determine how data communication and computer resources are assigned and allocated based on tasks requirements and their priorities in IoT environments.	<i>Chapter One: Introduction</i> <i>Chapter Two: Background and Related Work</i>
3. To identify a suitable methodology for building a resource allocation scheduler in fog computing framework for IoT environments.	<i>Chapter Three: Research Design and Methodology</i>
4. To build a resource allocation scheduler framework in fog computing for IoT environments.	<i>Chapter Four: Framework Design and Implementation</i>
5. To test and evaluate the proposed RAS in fog computing framework.	<i>Chapter Five: Results and Discussions</i>

6.4 Contributions of this Thesis

Two significant contributions made by this thesis can be categorized in theoretical terms and practical or experimental terms.

Firstly, **Chapter Two** of this research pinpointed the role played by fog computing by providing QoS in the existing systems. A review paper of Quality of Service in fog computing for the IoT was produced. To our knowledge with the comments received from the reviewers, this was the first review paper looking at QoS in fog computing. The paper was published in the “*International Journal for Fog Computing*” (Vambe *et al.*, 2020). The paper provided open challenges that are still faced in fog computing as far as improving QoS is concerned.

Still, in theoretical terms, another paper was submitted and accepted as a conference proceeding “<https://www.spu.ac.za/index.php/ieee-imatec-2020/>” and will be published in IEEE. The paper was based on the results of this study.

These papers act as a good starting point for other researchers in industry and academia in finding methods, procedures and strategies that can be implemented in fog computing. This is of paramount importance as it will help in improving fog computing as far as reducing round-trip time (latency) and improve QoS is concerned. In a nutshell, these two papers add to the board of knowledge as far as IoT-Fog-Cloud information is concerned.

As evidenced by our practical findings that RAS minimize round trip time, improve throughput, which leads to reduced latency and improved performance of the whole IoT-Fog-Cloud architecture, it can be implemented to existing systems. Having a system with a reduced round-trip time, improved QoS, meet the end-user expectations, promote faster communication and task execution plays a pivotal role in supporting smart environments. The introduced RAS will be able to support the emergence of the fourth industrial revolution, which is all about “smart world”. The RAS can also be implemented in smart homes, smart cities, smart health, smart industries and many smart IoT application areas. In smart homes, it would help in security issues as updates and alerts will be given in real-time on cases such as fire, theft, among other things that might cause a threat to the safety of humans.

In smart hospitals (health), patients will be monitored, their data analysed, and their databases updated in real-time, which will reduce problems usually caused by human errors. Situations that require real-time applications in smart health include monitoring of dialysis, heart problems, life support machines and many more. Human life will be lost if there is compromised round-trip time.

Cars and any faulty robots will be monitored in smart cities in real-time which will reduce accidents. Industries where artificial intelligence is taking a center stage through the use of robots which are used and controlled in doing jobs in real-time will also benefit from the benefits offered by RAS in fog computing.

This research is of paramount importance to South Africa as a country, Africa as a continent and the world in general as they are about to adopt the fourth industrial revolution to help our communities. In the South African context and Africa where there is low penetration of the internet infrastructure and the affordability issues of bandwidth, RAS will help in efficiently and effectively use the available bandwidth while minimizing latency in automated machines.

6.5 Future Work

Importantly, our results provide evidence that if resource allocation is done properly, round-trip time can be reduced and QoS can be improved in fog computing. However, future investigations are necessary where the approach will be tested in a real-world IoT environment to validate the conclusions that have been drawn from this study which are based on the simulation environment. To fully authenticate our findings, future research should consider the potential effects on queuing time, offloading time and throughput more carefully, for example when the RAS is implemented in a different setup where the IoT devices are mobile. Regardless of our findings, future research should continue to explore how roundtrip time can be minimized and QoS be improved in fog computing since many IoT devices are coming into play to fulfil the dream of living in a smart world where everything will be automated. As can be noticed, our research focused mainly on the IoT and fog layer. In other studies, the focus should also be given to the interaction between fog and cloud to continuously benefit from the finite computation, which can be harnessed on the cloud layer without compromising QoS.

In conclusion, as far as this research is concerned, it will be necessary that future researchers should also center their research based on the open research areas as highlighted in our literature review paper which includes: (a) Orchestration (Cloud-Fog) Challenge, (b) Computing Challenge and, (c) Management Challenges. These open research areas can also affect round-trip time and QoS. As such, there is a need to call for joint effort for further studies in fog computing and IoT-Fog-Cloud architecture.

6.6 Conclusions

This chapter articulated the summary of the whole thesis and highlighted how the research answered the aims, objectives and research questions. Of significant importance, the chapter explained the significance of this work and the contributions to the body of scientific knowledge of IoT, fog computing and cloud computing as far as round-trip time which affects latency and QoS is concerned. In conclusion, shortcomings and future research avenues have alluded.

References

Aazam, M. *et al.* (2016) 'MeFoRE: QoE based resource estimation at Fog to enhance QoS in IoT', in *2016 23rd International Conference on Telecommunications, ICT 2016*. doi: 10.1109/ICT.2016.7500362.

Aazam, M. and Huh, E. N. (2015a) 'Dynamic resource provisioning through Fog micro datacenter', in *2015 IEEE International Conference on Pervasive Computing and Communication Workshops, PerCom Workshops 2015*. doi: 10.1109/PERCOMW.2015.7134002.

Aazam, M. and Huh, E. N. (2015b) 'Fog computing micro datacenter based dynamic resource estimation and pricing model for IoT', in *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*. doi: 10.1109/AINA.2015.254.

Adhikari, M., Mukherjee, M. and Srirama, S. N. (2019) 'DPTO: A Deadline and Priority-aware Task Offloading in Fog Computing Framework Leveraging Multi-level Feedback Queueing', *IEEE Internet of Things Journal*. doi: 10.1109/jiot.2019.2946426.

Aissi, H., Bazgan, C. and Vanderpooten, D. (2005) 'Complexity of the min-max and min-max regret assignment problems', *Operations Research Letters*. doi: 10.1016/j.orl.2004.12.002.

Albishi, S. *et al.* (2017) 'Challenges and Solutions for Applications and Technologies in the Internet of Things', in *Procedia Computer Science*, pp. 608–614. doi: 10.1016/j.procs.2017.12.196.

Albishi, Saad *et al.* (2017) 'Challenges and Solutions for Applications and Technologies in the Internet of Things', in *Procedia Computer Science*, pp. 608–614. doi: 10.1016/j.procs.2017.12.196.

Alnoman, A. and Anpalagan, A. (2018) 'A Dynamic Priority Service Provision Scheme for Delay-Sensitive Applications in Fog Computing', *29th Biennial Symposium on Communications, BSC 2018*, (Bsc), pp. 1–5. doi: 10.1109/BSC.2018.8494691.

- Alsaffar, A. A. *et al.* (2017) 'An Architecture of Thin Client-Edge Computing Collaboration for Data Distribution and Resource Allocation in Cloud', (November).
- Aral, A. and Brandic, I. (2017) 'Quality of Service Channelling for Latency Sensitive Edge Applications', *Proceedings - 2017 IEEE 1st International Conference on Edge Computing, EDGE 2017*, (11 September), pp. 166–173. doi: 10.1109/IEEE.EDGE.2017.30.
- Artimy, M. M., Robertson, W. and Phillips, W. J. (2004) 'Connectivity in Inter-Vehicle Ad Hoc Networks', *Ccece 2004*, May. doi: 10.1109/CCECE.2004.1345014.
- Atlam, H. F., Walters, R. J. and Wills, G. B. (2018) 'Fog Computing and the Internet of Things: A Review', *Big Data and Cognitive Computing*, 10(April 2018). doi: 10.3390/bdcc2020010.
- Atzori, L., Iera, A. and Morabito, G. (2010) 'The Internet of Things: A survey', *Computer Networks*. doi: 10.1016/j.comnet.2010.05.010.
- Baun, C., Kunze, M., Nimis, J., Tai, S., *et al.* (2011) 'Cloud Basics', in *Cloud Computing*. doi: 10.1007/978-3-642-20917-8_2.
- Baun, C., Kunze, M., Nimis, J. and Tai, S. (2011) *Cloud Computing Web-basierte dynamische IT-Services, Web-basierte dynamische IT-Services*. doi: 10.1007/978-3-642-18436-9.
- Beheshti, Z. and Shamsuddin, S. M. H. (2013) 'A review of population-based meta-heuristic algorithm', *International Journal of Advances in Soft Computing and its Applications*.
- Beloglazov, A. *et al.* (2011) *A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems*, *Advances in Computers*. doi: 10.1016/B978-0-12-385512-1.00003-7.
- Biljana, L. R. S. and Kire, V. T. (2016) 'A review of Internet of Things for smart home: Challenges and solutions', *Journal of Cleaner Production*.
- Bittencourt, L. F. *et al.* (2016) 'Towards Virtual Machine Migration in Fog Computing', *Proceedings - 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2015*, (03 March), pp. 1–8. doi: 10.1109/3PGCIC.2015.85.

- Bonomi, F. *et al.* (2012) 'Fog Computing and Its Role in the Internet of Things', *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, (August 17, 2012), pp. 13–16. doi: 10.1145/2342509.2342513.
- Bonomi, F. *et al.* (2014) 'Fog computing: A platform for internet of things and analytics', *Studies in Computational Intelligence*. doi: 10.1007/978-3-319-05029-4_7.
- Botta, A. *et al.* (2016) 'Integration of Cloud computing and Internet of Things: A survey', *Future Generation Computer Systems*, 56(October 2015), pp. 684–700. doi: 10.1016/j.future.2015.09.021.
- Braun, T. D. *et al.* (2001) 'A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems', *Journal of Parallel and Distributed Computing*. doi: 10.1006/jpdc.2000.1714.
- Calheiros, R. N. *et al.* (2011) 'CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms', *Software - Practice and Experience*. doi: 10.1002/spe.995.
- Caposelle, A. T. *et al.* (2016) 'Counteracting denial-of-sleep attacks in wake-up-radio-based sensing systems', in *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking, SECON 2016*, p. 03 November 2016. doi: 10.1109/SAHCN.2016.7732978.
- Chaisiri, S., Lee, B. B. S. and Niyato, D. (2012) 'Optimization of resource provisioning cost in cloud computing', *Services Computing, IEEE* doi: 10.1109/TSC.2011.7.
- Chang, C., Srirama, S. N. and Buyya, R. (2016) 'Mobile Cloud Business Process Management System for the internet of things: A survey', *ACM Computing Surveys*, 49(4). doi: 10.1145/3012000.
- Chang, C., Srirama, S. N. and Buyya, R. (2017) 'Internet of Things (IoT) and New Computing Paradigms', in *Fog and Edge Computing: Principles and Paradigms*, pp. 1–23. Available at: www.buyya.com/papers/C01_Introduction_FEC.pdf%0A.
- Chekired, D. A. and Khoukhi, L. (2018) 'Multi-tier fog architecture: A new delay-tolerant network for iot data processing', in *IEEE International Conference on Communications*. doi: 10.1109/ICC.2018.8422170.

- Chen, Z. *et al.* (2017) 'An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance', *Proceedings of the Second ACM/IEEE Symposium on Edge Computing - SEC '17*, October, pp. 1–14. doi: 10.1145/3132211.3134458.
- Choudhari, T., Moh, M. and Moh, T. S. (2018) 'Prioritized task scheduling in fog computing', in *Proceedings of the ACMSE 2018 Conference*. doi: 10.1145/3190645.3190699.
- Chowdhury, M., ... E. S.-I. T. on and 2018, U. (2018) 'Context-Aware Task Migration for HART-Centric Collaboration over FiWi Based Tactile Internet Infrastructures', *IEEE Explore.Ieee.Org*, 29, June(6), pp. 1231–1246. Available at: <https://ieeexplore.ieee.org/iel7/71/4359390/08252736.pdf>.
- Cisco (2014) 'Quality of Service Overview', *Cisco IOS Quality of Service Solutions Configuration Guide*, (January 30), p. 18. Available at: https://www.cisco.com/c/en/us/td/docs/ios/12_2/qos/configuration/guide/fqos_c/qcfintro.pdf.
- 'Cloud computing: Web-based dynamic IT services' (2012) *Choice Reviews Online*. doi: 10.5860/choice.49-2712.
- Coutinho, A. A. T. R. ., Greve, F. and Prazeres, C. (2017) 'An Architecture for Fog Computing Emulation', *Wcga - Sbrc*, 15(1/2017). Available at: <http://ojs.sbc.org.br/index.php/wcga/article/view/2552>.
- Daj, A., Samoilă, C. and Ursuțiu, D. (2012) 'Digital marketing and regulatory challenges of Machine-to-Machine (M2M) communications', in *2012 9th International Conference on Remote Engineering and Virtual Instrumentation, REV 2012*, p. 05 September 2012. doi: 10.1109/REV.2012.6293118.
- Dana Jošilo, S. and Dán, G. (2018) 'Decentralized Algorithm for Randomized Task Allocation in Fog Computing Systems', 18 May. Available at: <http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1206706&dswid=-5375>.
- Dao, N.-N. *et al.* (2018) 'Pattern-Identified Online Task Scheduling in Multitier Edge Computing for Industrial IoT Services', *Mobile Information Systems*, 2018, 04 A. doi: 10.1155/2018/2101206.

- Dastjerdi, A. V. and Buyya, R. (2016) 'Fog Computing: Helping the Internet of Things Realize Its Potential', *Computer*, 49(8, August), pp. 112–116. doi: 10.1109/MC.2016.245.
- Dilworth, I. J. (2012) 'Bluetooth', in *The Cable and Telecommunications Professionals' Reference: PSTN, IP and Cellular Networks, and Mathematical Techniques*. doi: 10.4324/9780080475189-23.
- Dinh, H. T. *et al.* (2013) 'A survey of mobile cloud computing: Architecture, applications, and approaches', *Wireless Communications and Mobile Computing*. doi: 10.1002/wcm.1203.
- Ericsson (2011) 'More Than 50 Billion Connected Devices', *White Paper*, (February), pp. 1–12. doi: 284 23-3149 Uen.
- Fan, Q. and Ansari, N. (2018) 'Towards Workload Balancing in Fog Computing Empowered IoT', *IEEE Transactions on Network Science and Engineering*. doi: 10.1109/TNSE.2018.2852762.
- Gia, T. N. *et al.* (2018) 'Fog computing approach for mobility support in internet-of-things systems', *IEEE Access*, 6(June), pp. 36064–36082. doi: 10.1109/ACCESS.2018.2848119.
- Giovanni and Surantha, N. (2018) 'Design and Evaluation of Enterprise Network with Converged Services', in *Procedia Computer Science*. doi: 10.1016/j.procs.2018.08.205.
- Google Press (2006) 'Conversation with Eric Schmidt hosted by Danny Sullivan', in Danny, S. (ed.). Google Press Centre. Available at: <https://www.google.com/press/podium/ses2006.html>.
- Gubbi, J. *et al.* (2013) 'Internet of Things (IoT): A vision, architectural elements, and future directions', *Future Generation Computer Systems*. doi: 10.1016/j.future.2013.01.010.
- Gupta, H. *et al.* (2017) 'iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments', in *Software - Practice and Experience*. doi: 10.1002/spe.2509.
- He, X. *et al.* (2016) 'A novel load balancing strategy of software-defined cloud/fog

networking in the Internet of Vehicles', *China Communications*, 13, pp. 140–149. doi: 10.1109/CC.2016.7833468.

He, Y., Zhao, N. and Yin, H. (2018) 'Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach', *IEEE Transactions on Vehicular Technology*, 67-January(1), pp. 44–55. doi: 10.1109/TVT.2017.2760281.

Higashino, W. A., Capretz, M. A. M. and Bittencourt, L. F. (2016) 'CEPSim: Modelling and simulation of Complex Event Processing systems in cloud environments', *Future Generation Computer Systems*. doi: 10.1016/j.future.2015.10.023.

Iorga, M., Martin, M. J. and Feldman, L. (2018) 'Fog Computing Conceptual Model NIST Special Publication 500-325', (14 March). Available at: <https://www.nist.gov/publications/fog-computing-conceptual-model>.

Iotti, N. *et al.* (2017) 'Improving Quality of Experience in Future Wireless Access Networks through Fog Computing', *IEEE Internet Computing*, 21-March/(2), pp. 26–33. doi: 10.1109/MIC.2017.38.

Islam, S. M. R. *et al.* (2015) 'The Internet of Things for Health Care : A Comprehensive Survey', *Access, IEEE*, 3(01 June 2015), pp. 678–708. doi: 10.1109/ACCESS.2015.2437951.

Jain Kansal, N. and Chana, I. (2012) 'Cloud Load Balancing Techniques : A Step Towards Green Computing', *IJCSI International Journal of Computer Science Issues*, Vol. 9, Ja(Issue 1), pp. 238–246. doi: 10.1145/2382756.2382784.

Jim Turley (2014) 'Introduction to Intel R Architecture - The Basics.', (White Paper, Introduction to Intel), p. 10. Available at: <http://ark.intel.com>.

Kalyvianaki, E. (2008) *Resource provisioning for virtualized server applications*, *Pediatrics*.

Keller, G. *et al.* (2012) 'An analysis of first fit heuristics for the virtual machine relocation problem', in *Proceedings of the 2012 8th International Conference on Network and Service Management, CNSM 2012*.

Kiani, A. and Ansari, N. (2018) 'Edge Computing Aware NOMA for 5G Networks',

- IEEE Internet of Things Journal*, 5-April(2), pp. 1299–1306. doi: 10.1109/JIOT.2018.2796542.
- Kim, J. and Lee, J. W. (2014) 'OpenIoT: An open service framework for the Internet of Things', in *2014 IEEE World Forum on Internet of Things, WF-IoT 2014*. doi: 10.1109/WF-IoT.2014.6803126.
- Ko, S. W. *et al.* (2017) 'Live Prefetching for Mobile Computation Offloading', *IEEE Transactions on Wireless Communications*. doi: 10.1109/TWC.2017.2674665.
- Kochovski, P. and Stankovski, V. (2018) 'Supporting smart construction with dependable edge computing infrastructures and applications', *Automation in Construction*, 85(May 2017), pp. 182–192. doi: 10.1016/j.autcon.2017.10.008.
- Li, C. *et al.* (2018) 'Edge-Oriented Computing Paradigms : A Survey on Architecture Design and System Management', 51(2, June 02 2018).
- Li, G. *et al.* (2018) 'Method of Resource Estimation Based on QoS in Edge Computing', 2018(December 31).
- Li, J. *et al.* (2017) 'Resource Management in Fog-Enhanced Radio Access Network to Support Real-Time Vehicular Services', in *Proceedings - 2017 IEEE 1st International Conference on Fog and Edge Computing, ICFEC 2017*, pp. 68–74. doi: 10.1109/ICFEC.2017.17.
- Liu, C. F. *et al.* (2019) 'Dynamic Task Offloading and Resource Allocation for Ultra-Reliable Low-Latency Edge Computing', in *IEEE Transactions on Communications*. doi: 10.1109/TCOMM.2019.2898573.
- Liu, L. *et al.* (2018) 'A Task Scheduling Algorithm Based on Classification Mining in Fog Computing Environment', *Wireless Communications and Mobile Computing*, 2018, Augu. doi: 10.1155/2018/2102348.
- Liu, Y. *et al.* (2017) 'Incentive mechanism for computation offloading using edge computing: A Stackelberg game approach', *Computer Networks*, 129(December 24), pp. 399–409. doi: 10.1016/j.comnet.2017.03.015.
- Lu, Z. *et al.* (2015) 'Task allocation for mobile cloud computing in heterogeneous wireless networks', in *Proceedings - International Conference on Computer Communications and Networks, ICCCN*. doi: 10.1109/ICCCN.2015.7288473.

- Luan, T. H. *et al.* (2015) 'Fog Computing: Focusing on Mobile Users at the Edge', (arXiv: 1502.01815v1 [cs.NI] 6 February). doi: 10.1016/j.jnca.2015.02.002.
- Lv, Y. *et al.* (2015) 'Traffic Flow Prediction with Big Data: A Deep Learning Approach', *IEEE Transactions on Intelligent Transportation Systems*, 16(April 2015). doi: 10.1109/TITS.2014.2345663.
- Madni, S. H. H. *et al.* (2017) 'Performance comparison of heuristic algorithms for task scheduling in IaaS cloud computing environment', *PLoS ONE*. doi: 10.1371/journal.pone.0176321.
- Maheswaran, M. *et al.* (1999) 'Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems', *Proceedings of the Heterogeneous Computing Workshop, HCW*. doi: 10.1109/hcw.1999.765094.
- Mahmud, M. R. *et al.* (2016) 'Maximizing quality of experience through context-aware mobile application scheduling in cloudlet infrastructure', *Software - Practice and Experience*, 46(11, 29 December), pp. 1525–1545. doi: 10.1002/spe.2392.
- Micrac, I. (2008) *Internet of Things in 2020, Roadmap for future*, European Commission, Information Society and Media.
- Mtshali, M. *et al.* (2019) 'Multi-objective optimization approach for task scheduling in fog computing', in *icABCD 2019 - 2nd International Conference on Advances in Big Data, Computing and Data Communication Systems*. doi: 10.1109/ICABCD.2019.8851038.
- Mukherjee, M. *et al.* (2018) 'Transmission and Latency-Aware Load Balancing for Fog Radio Access Networks', in *2018 IEEE Global Communications Conference, GLOBECOM 2018 - Proceedings*. doi: 10.1109/GLOCOM.2018.8647580.
- Mukherjee, M. *et al.* (2019) 'Joint Task Offloading and Resource Allocation for Delay-Sensitive Fog Networks', in *IEEE International Conference on Communications*. doi: 10.1109/ICC.2019.8761239.
- Mukherjee, M. *et al.* (2020) 'Latency-Driven Parallel Task Data Offloading in Fog Computing Networks for Industrial Applications', *IEEE Transactions on Industrial Informatics*. doi: 10.1109/TII.2019.2957129.
- Müller-Merbach, H. (1981) 'Heuristics and their design: a survey', *European Journal*

of *Operational Research*. doi: 10.1016/0377-2217(81)90024-2.

Müller-Merbach, H. (1985) 'Heuristics: Intelligent search strategies for computer problem solving', *European Journal of Operational Research*. doi: 10.1016/0377-2217(85)90047-5.

Mulyawan, B. (2011) 'Campus Network Design And Implementation Using Top Down Approach : A Case Study Tarumanagara University', *International Conference on Information Systems For Business Competitiveness*.

Nachiket (2019) *Smart Home Devices Market Scope by Trends and Opportunities to Expand Significantly by 2028, Market Industry Reports*. Available at: <https://aindustryreports.com/2019/05/21/smart-home-devices-market-scope-by-trends-and-opportunities-to-expand-significantly-by-2028/> (Accessed: 21 May 2019).

Naha, R. K. *et al.* (2018) 'Fog computing: Survey of trends, architectures, requirements, and research directions', *IEEE Access*. doi: 10.1109/ACCESS.2018.2866491.

Nath, S. B. *et al.* (2018) 'A Survey of Fog Computing and Communication: Current Researches and Future Directions', (arXiv:1804.04365v1 [cs.NI] 12 Apr 2018), pp. 1–47. Available at: <http://arxiv.org/abs/1804.04365>.

Neto, E. C. P., Callou, G. and Aires, F. (2017) 'An algorithm to optimise the load distribution of fog environments', *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, (01 December), pp. 1292–1297. doi: 10.1109/SMC.2017.8122791.

Newman, S. (2015) *Building Microservices*, O'Reilly.

Ni, L., Zhang, J. and Yu, J. (2018) 'Priced timed petri nets based resource allocation strategy for fog computing', in *Proceedings - 2016 International Conference on Identification, Information and Knowledge in the Internet of Things, IIKI 2016*, pp. 39–44. doi: 10.1109/IIKI.2016.87.

OpenFog Consortium Architecture Working Group (2017) '12 - OpenFog Reference Architecture for Fog Computing', *OpenFogConsortium*, (February), pp. 1–162. doi: OPFRA001.020817.

ORACLE *et al.* (2014) *Capa de aplicación (Guía de administración del sistema:*

servicios IP), *Revista Tecnura*. Available at:
https://www.academia.edu/8893403/METODOLOGIAS_PARA_EL_DISEÑO_DE_REDES_Contentido?auto=download%5Cnhttp://revistas.udistrital.edu.co/ojs/index.php/Tecnura/article/view/6901/8509%5Cnhttp://www.valleytalk.org/wp-content/uploads/2013/01/top-down-network-design.

Osanaiye, O. *et al.* (2017) 'From Cloud to Fog Computing: A Review and a Conceptual Live VM Migration Framework', *IEEE Access*, 5(11 April 2017), pp. 8284–8300. doi: 10.1109/ACCESS.2017.2692960.

Patel, K. K., Patel, S. M. and Scholar, P. G. (2016) 'Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges', *International Journal of Engineering Science and Computing*, 6(5), pp. 1–10. doi: 10.4010/2016.1482.

Pautasso, C., Zimmermann, O. and Leymann, F. (2008) 'RESTfulWeb Services vs. "Big"Web Services: Making the Right Architectural Decision', in *Proceedings of the 17th World Wide Web Conference*,. doi: 10.1145/1367497.1367606.

Perera, C., Liu, C. H. I. H., *et al.* (2014) 'A Survey on Internet of Things From Industrial Market Perspective', *IEEE Access*, 2(January 26,), pp. 1660–1679. doi: 10.1109/ACCESS.2015.2389854.

Perera, C., Zaslavsky, A., *et al.* (2014) 'Context aware computing for the internet of things: A survey', *IEEE Communications Surveys and Tutorials*, 16(1, 03 May), pp. 414–454. doi: 10.1109/SURV.2013.042313.00197.

Pérez, J. L. *et al.* (2018) 'A resilient and distributed near real-time traffic forecasting application for Fog computing environments', *Future Generation Computer Systems*, 87, Octobe. doi: 10.1016/j.future.2018.05.013.

Pham, X. Q. *et al.* (2017) 'A cost- and performance-effective approach for task scheduling based on collaboration between cloud and fog computing', *International Journal of Distributed Sensor Networks*, 13–11 Oct(11). doi: 10.1177/1550147717742073.

Rashidi, S. and Sharifian, S. (2017) 'Cloudlet dynamic server selection policy for mobile task off-loading in mobile cloud computing using soft computing techniques',

Journal of Supercomputing, 73(9, February), pp. 3796–3820. doi: 10.1007/s11227-017-1983-0.

Rasmila, R. and Laksana, T. G. (2019) 'The Implementation of Top Down Approach Method on Redesign of LAN Harvani Hotel Palembang', *JURNAL INFOTEL*. doi: 10.20895/infotel.v11i1.410.

Riehle, D. (2000) *Framework Design: A Role Modeling Approach*, *Design*. doi: 10.3929/ethz-a-003867001.

Rothlauf, F. (2011) *Design of Modern Heuristics: principles and Application*, *Design of Modern Heuristics: Principles and Application*.

Saad, M. (2018) 'Fog Computing and Its Role in the Internet of Things : Concept , Security and Privacy Issues', *International Journal of Computer Applications*, 180(32, April 2018), pp. 7–9.

Sampei, S. (2017) 'Development of wireless access and flexible networking technologies for 5G cellular systems', *IEICE Transactions on Communications*, E100B(8, August 8), pp. 1174–1180. doi: 10.1587/transcom.2016FGI0001.

Satria, D., Park, D. and Jo, M. (2017) 'Recovery for overloaded mobile edge computing', *Future Generation Computer Systems*, 70(14 July 2016), pp. 138–147. doi: 10.1016/j.future.2016.06.024.

Sensor, W. (2009) *WIRELESS SENSOR NETWORKS A Networking, Zywienie Czlowieka I Metabolizm*. doi: 10.1109/IEMBS.2008.4650376.

Shen, H. *et al.* (2015) 'C2EM: cloud-assisted complex event monitoring in wireless multimedia sensor networks', *Eurasip Journal on Wireless Communications and Networking*, 2015(1, April). doi: 10.1186/s13638-015-0347-9.

Shukla, S. *et al.* (2019) 'An analytical model to minimize the latency in healthcare internet-of-things in fog computing environment', *PLoS ONE*. doi: 10.1371/journal.pone.0224934.

Simonet, A., Lebre, A. and Orgerie, A. C. (2016) 'Deploying distributed cloud infrastructures: Who and at what cost?', in *Proceedings - 2016 IEEE International Conference on Cloud Engineering Workshops, IC2EW 2016*, pp. 178-183,04 August 2016. doi: 10.1109/IC2EW.2016.48.

- Singh, S. and Chana, I. (2015) 'QoS-aware autonomic resource management in cloud computing: A systematic review', *ACM Computing Surveys*. doi: 10.1145/2843889.
- Skarlat, O. *et al.* (2016) 'Resource provisioning for IoT services in the fog', *Proceedings - 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications, SOCA 2016*, (26 December 2016), pp. 32–39. doi: 10.1109/SOCA.2016.10.
- Skarlat, O., Nardelli, M., Schulte, S., Borkowski, M., *et al.* (2017) 'Optimized IoT service placement in the fog', *Service Oriented Computing and Applications*, 11(4, October 04), pp. 427–443. doi: 10.1007/s11761-017-0219-8.
- Skarlat, O., Nardelli, M., Schulte, S. and Dustdar, S. (2017) 'Towards QoS-Aware Fog Service Placement', *Proceedings - 2017 IEEE 1st International Conference on Fog and Edge Computing, ICFEC 2017*, (24 August), pp. 89–96. doi: 10.1109/ICFEC.2017.12.
- Soltész, S. *et al.* (2007) 'Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors', in *Operating Systems Review (ACM)*. doi: 10.1145/1272996.1273025.
- Song, Y. *et al.* (2017) 'An Approach to QoS-based Task Distribution in Edge Computing Networks for IoT Applications', *Proceedings - 2017 IEEE 1st International Conference on Edge Computing, EDGE 2017*, (11 September), pp. 32–39. doi: 10.1109/IEEE.EDGE.2017.50.
- Souza, V. B. *et al.* (2017) 'Towards a Fog-to-Cloud control topology for QoS-aware end-to-end communication', in *2017 IEEE/ACM 25th International Symposium on Quality of Service, IWQoS 2017*. doi: 10.1109/IWQoS.2017.7969140.
- Taleb, T. *et al.* (2017) 'Mobile edge computing potential in making cities smarter', *IEEE Communications Magazine*, 55(3, March 13), pp. 38–43. doi: 10.1109/MCOM.2017.1600249CM.
- Taneja, M. and Davy, A. (2017) 'Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm', *Proceedings of the IM 2017 - 2017 IFIP/IEEE International Symposium on Integrated Network and Service*

Management, (24 July), pp. 1222–1228. doi: 10.23919/INM.2017.7987464.

The National Institute of Standards and Technology (2011) 'The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology', *NIST Special Publication*. doi: 10.1136/emj.2010.096966.

Tran, T. X. *et al.* (2017) 'Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges', *IEEE Communications Magazine*. doi: 10.1109/MCOM.2017.1600863.

Vambe, W. T., Chang, C. and Sibanda, K. (2020) 'A Review of Quality of Service in Fog Computing for the Internet of Things', 3(1), pp. 22–40. doi: 10.4018/IJFC.2020010102.

Vaquero, L. M. and Rodero-Merino, L. (2014) 'Finding your Way in the Fog: Towards a Comprehensive Definition of Fog Computing', *ACM SIGCOMM Computer Communication Review*, 44(5, 10 October), pp. 27–32. doi: 10.1145/2677046.2677052.

Vermesan, O. and Friess, P. (2014) *Internet of things applications: From research and innovation to market deployment*, *Internet of Things Applications: From Research and Innovation to Market Deployment*.

Vögler, M. *et al.* (2016) 'A Scalable Framework for Provisioning Large-Scale IoT Deployments', *ACM Transactions on Internet Technology*. doi: 10.1145/2850416.

Wang, C. *et al.* (2016) 'On the Serviceability of Mobile Vehicular Cloudlets in a Large-Scale Urban Environment', *IEEE Transactions on Intelligent Transportation Systems*, 17, October(10), pp. 2960–2970. doi: 10.1109/TITS.2016.2561293.

Wang, C., Gill, C. and Lu, C. (2017) 'Real-time middleware for cyber-physical event processing', *2017 IEEE/ACM 25th International Symposium on Quality of Service, IWQoS 2017*, (07 July). doi: 10.1109/IWQoS.2017.7969159.

Wang, K. and Yang, K. (2017) 'Power-minimization computing resource allocation in mobile cloud-radio access network', *Proceedings - 2016 16th IEEE International Conference on Computer and Information Technology, CIT 2016, 2016 6th International Symposium on Cloud and Service Computing, IEEE SC2 2016 and 2016 International Symposium on Security and Privacy in Social Netwo*, (13 March),

pp. 667–672. doi: 10.1109/CIT.2016.64.

Wang, S. *et al.* (2018) 'A Survey on Service Migration in Mobile Edge Computing', *IEEE Access*, 6, pp. 23511–23528. doi: 10.1109/ACCESS.2018.2828102.

Wang, Y. *et al.* (2019) 'Cooperative Task Offloading in Three-Tier Mobile Computing Networks: An ADMM Framework', *IEEE Transactions on Vehicular Technology*. doi: 10.1109/TVT.2019.2892176.

Wang, Z. *et al.* (2018) 'User mobility aware task assignment for Mobile Edge Computing', *Future Generation Computer Systems*, 85, March, pp. 1–8. doi: 10.1016/j.future.2018.02.014.

Xiao, Y. and Krunz, M. (2017) 'QoE and power efficiency tradeoff for fog computing networks with fog node cooperation', *Proceedings - IEEE INFOCOM*, (May). doi: 10.1109/INFOCOM.2017.8057196.

Xu, Y. and Helal, S. (2014) 'Application caching for cloud-sensor systems', in *Proceedings of the 17th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems - MSWiM '14*, pp. 303–306, September 21. doi: 10.1145/2641798.2641814.

Yang, B. *et al.* (2018) 'Cost-Efficient NFV-Enabled Mobile Edge-Cloud for Low Latency Mobile Applications', *IEEE Transactions on Network and Service Management*, 15(1), pp. 475–488. doi: 10.1109/TNSM.2018.2790081.

Yang, X. and Rahmani, N. (2020) 'Task scheduling mechanisms in fog computing: review, trends, and perspectives', *Kybernetes*. doi: 10.1108/K-10-2019-0666.

Yang, Y. *et al.* (2019) 'POMT: Paired Offloading of Multiple Tasks in Heterogeneous Fog Networks', *IEEE Internet of Things Journal*. doi: 10.1109/JIOT.2019.2922324.

Yao, H. *et al.* (2017) 'Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing', in *Concurrency Computation*. doi: 10.1002/cpe.3975.

Yi, S. *et al.* (2016) 'Fog computing: Platform and applications', *Proceedings - 3rd Workshop on Hot Topics in Web Systems and Technologies, HotWeb 2015*, pp. 73–78. doi: 10.1109/HotWeb.2015.22.

- Yi, S., Li, C. and Li, Q. (2015) 'A Survey of Fog Computing: Concepts, Applications and Issues', *Proceedings of the 2015 Workshop on Mobile Big Data - Mobidata '15*, (15, June 21,), pp. 37–42. doi: 10.1145/2757384.2757397.
- Yousefpour, A. *et al.* (2018) 'On Reducing IoT Service Delay via Fog Offloading', *IEEE Internet of Things Journal*, 5, April(2), pp. 998–1010. doi: 10.1109/JIOT.2017.2788802.
- Zhan, Z.-H. *et al.* (2015) 'Cloud Computing Resource Scheduling and a Survey of Its Evolutionary Approaches', *ACM Computing Surveys*, 47, July(4), pp. 1–33. doi: 10.1145/2788397.
- Zhang, G. *et al.* (2019) 'FEMTO: Fair and energy-minimized task offloading for fog-enabled IoT networks', *IEEE Internet of Things Journal*. doi: 10.1109/JIOT.2018.2887229.
- Zhang, X. and Zhu, Q. (2017) 'Statistical Quality of Service Provisioning Over Edge Computing Mobile Wireless Networks', (1,11 December), pp. 412–417.
- Zhao, L. *et al.* (2018) 'Optimal Placement of Cloudlets for Access Delay Minimization in SDN-based Internet of Things Networks', *IEEE Internet of Things Journal*, 5(2, April), pp. 1–12. doi: 10.1109/JIOT.2018.2811808.
- Zheng, X., Xu, L. Da and Chai, S. (2017) 'QoS Recommendation in Cloud Services', *IEEE Access*, 5(19 April), pp. 5171–5177. doi: 10.1109/ACCESS.2017.2695657.

International Journal of Fog Computing
Volume 3 • Issue 1 • January-June 2020

A Review of Quality of Service in Fog Computing for the Internet of Things

William Tichaona Vambe, University of Fort Hare, Alice, South Africa

 <https://orcid.org/0000-0003-0516-1260>

Chii Chang, University of Melbourne, Melbourne, Australia

Khulumani Sibanda, University of Fort Hare, Alice, South Africa

ABSTRACT

With the advent of the paradigm of the Internet of Things, many computing elements need many modifications to promote Quality of Service (QoS). Quality of Service is a pillar that promotes real-time reaction to time-critical tasks. Any impediments to QoS should be resolved and handled. In 2012, fog computing was implemented to enhance QoS in current systems in a bid to tackle QoS problems encountered by using cloud computing alone. Currently, the primary focus in fog computing is now on enhancing QoS. The primary goal of this study is, therefore, to critically review and evaluate the literature on the work done to improve elements of QoS in fog computing. This study begins by examining the roots of history, characteristics, and advantages of fog computing. Secondly, it discusses the important elements of QoS parameters. Finally, open problems that still affect fog computing are identified and discussed in order to achieve enhanced QoS.

INTRODUCTION

The Internet of Things (IoT) is defined as a vibrant worldwide data network composed of internet-connected objects such as radio-frequency identifiers, sensors, and actuators, as well as other devices and smart devices that are becoming an essential part of the Internet (Perera, Liu, Jayawardena, & Chen, 2014). The word IoT can be traced back to the early 1990s when Kelvin Ashton introduced it (S. Albishi, Soh, Ullah, & Algarni, 2017). Over the years, IoT has received considerable attention due to the capacity to interact and execute some tasks together or react to incidents without specific instructions (Perera, Zaslavsky, Christen, & Georgakopoulos, 2014). Intelligence, Connectivity, Dynamic Scale, Enormous Nature, Sensing, Heterogeneity, and Security are the key fundamental characteristics which drive IoT (Ericsson, 2011). The above-mentioned features have contributed considerably to the successful adoption plus the use of IoT in current information systems and applications, creating value and support for human operations (Perera, Liu, et al., 2014). Collected works demonstrate that IoT has been implemented in various fields, leading to the development of smart cities, intelligent energy, and electrical grids, intelligent homes, smart buildings and infrastructure, intelligent health just to mention a few (Saad et al., 2017).

DOI: 10.4018/IJFC.2020010102

Copyright©2020,IGIGlobal.Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

This “smart world” has changed the manner in which people live and work by saving time and organizational resources whilst bringing new opportunities for knowledge formation, innovation, and development (Capossele, Cervo, Petrioli, & Spenza, 2016).

After the realization that the “things” that make up the IoT ecosystem have limited processing power and storage, cloud computing was introduced and integrated into IoT to provide scalable storage and processing services to meet IoT demands (Atlam, Walters, & Wills, 2018). In spite of cloud computing advantages in terms of storage and processing services, it still suffers mostly in providing low latency (Satria, Park, & Jo, 2017). This is because of its geographical location to the devices it wants to offer services. High latency compromise QoS which cause communication delays due to unstable and intermittent network connectivity. Explicitly, the unprecedented amount of data produced by IoT devices (Dastjerdi & Buyya, 2016) burden the network resulting in network transmission delays. Additionally, sending such huge data to and from the cloud requires exceptionally high network bandwidth (Atlam et al., 2018). With the anticipated 50 billion intelligent interconnected device deployments serving various vertical markets by 2020, QoS is probable to be compromised which in turn affect time-sensitive functions which have been backed by cloud computing. As such, this has triggered a concerted effort to come with adaptive and decentralized computational paradigms that complement the centralized cloud computing model serving IoT networks. To fill this technological gap, new concepts and technologies have been developed to manage this growing fleet of IoT devices. Specifically, fog computing which was introduced by Cisco has gained much attention (Bonomi, Milito, Zhu, & Addepalli, 2012). OpenFog Consortium (the IEEE standard) defined fog computing as “*a horizontal, system-level architecture that distributes computing, storage, control, and networking functions closer to the users along a cloud-to-thing continuum*” (OpenFog Consortium Architecture Working Group, 2017). Fog computing architecture consists of fog (physical or virtual), residing between smart end-devices and centralized (cloud) services which facilitates minimization of the request-response time from-to supported applications, and provides, for the end-devices, local computing resources and, when needed, network connectivity to centralized services (Iorga, Martin, & Feldman, 2018). These are achieved through fog computing ability to support: i) Low latency and location awareness; ii) Extensive geographical dispersal; iii) Mobility; iv) Very large number of nodes, v) Predominant role of wireless access, vi) Strong presence of streaming and real-time applications, vii) Heterogeneity, thus supporting critical IoT services and applications to have improved QoS (Atlam et al., 2018).

Since its inception in 2012, fog computing has gained much attention in both academic and industrial space because of its advantages in supporting the Internet of Things technologies and providing improved QoS. Several surveys whose main topics cover fog computing key features (Vaquero & Roderio-Merino, 2014), platform and paradigm (Xu & Helal, 2014), architecture design (Simonet, Lebre, & Orgerie, 2016), security, and privacy (Osanaiye et al., 2017) has been done and in-depth. However, to the best of our knowledge, there are no existing related survey papers of fog computing whose main perspective is on QoS. The primary purpose of this study is, therefore, to review and critically evaluate current literature on the work that has been done to tackle difficulties and enhance QoS elements in fog computing. Conclusively, open researches areas and future re-scopes for QoS of fog computing will be underscored.

BACKGROUND

Providing satisfactory QoS is a fundamental goal in networking, cloud services or in general information systems. Depending on the perspective, QoS can have several definitions. From a

networking perspective, QoS refers to any technology that manages data traffic to reduce packet loss, latency and jitter on the network (Cisco, 2014). In general information systems, Quality of Service is the capacity to prioritize distinct applications, customers or information flows or to ensure a certain level of information stream efficiency. In cloud computing, QoS is “*non-functional properties of cloud services, which describe how well a service is performed, such as compliance, availability, reliability, responsiveness, price, security, latency, etc.*” (Zheng, Xu, & Chai, 2017). The major parameters that define QoS include throughput, transit delay, availability, priority, jitter, etc. In the light of the definitions above, QoS is very important because it promotes improved services.

Correspondingly, studies have been done both in academia and industrial domain on how fog computing technologies can be used and implemented in existing systems to enhance QoS. Failure to maintain high QoS in fog computing have a negative impact on fog computing-based systems and or applications. This will cause fog computing-based systems and applications to encounter end-to-end communication delays(Souza et al., 2017), service migration issues (Song et al., 2017), workload deployment challenges (Taneja & Davy, 2017), computation and resource allocation problems (Chao et al., 2017), etc. For this reason, it is wise to continuously devise ways and strategies to maintain high QoS in fog computing as to avoid the above-mentioned problems if we are to successfully meet and support demands of the dynamic IoT ecosystem.

As affirmed from prior existing research, work which covers fog computing key features, paradigm, design, security, and privacy has been done as main topics and in-depth. In contrast, none has looked at QoS in fog computing as the main topic and in detail. The work that exists, for example (Yi, Li, & Li, 2015) focused on identifying problem domains of fog computing. In a synopsis, Yi et al (2015) looked at the QoS and summarized four aspects of fog service which help to achieve QoS namely connectivity, reliability, capacity, and delay. Researchers Li et al. did a survey which looked at architecture design and system management. The work described major optimization solutions and heuristic approaches to deal with time transmission, execution time, round-trip time, real-time support to address delay and execution time(C. Li et al., 2018). Researchers, Nath et al (2018) did a survey whose focus was on fog computing system architectures, fog enabling technologies and features, privacy and security of fog, the QoS parameters, and application of fog (Nath et al., 2018). Even though these survey papers have mentioned QoS, they did not specifically go in deepness. Since QoS is an important factor in fog computing for IoT especially for time-critical application, this paper serves as a complement survey paper to the research field of fog computing.

The survey is structured as follows. In the next section, the authors start by describing and defining the meanings of the aspects of QoS and the problems or challenges they cause in fulfilling QoS requirements in fog computing. Furthermore, a look in the literature on how researchers addressed the problems and how they have improved QoS aspects in fog computing. In conclusion, open research areas are highlighted, discussed as areas that need attention and further research.

QUALITY OF SERVICE IN FOG COMPUTING

In order to get enhanced QoS, fog computing technology has been developed and implemented. Without any doubt, as supported by the literature, fog computing has managed to offer improved QoS among other advantages. Several systems and applications have adopted the use of fog technology over the years to improve QoS in the existing systems. Quality of Service can be classified into the following aspects (see **Figure 8: QoS aspect taxonomy tree** below). Much

attention has been paid to enhancing QoS in fog computing recently. Several approaches have been suggested and implemented to address QoS aspects challenges and positive strides have been made in that regard.

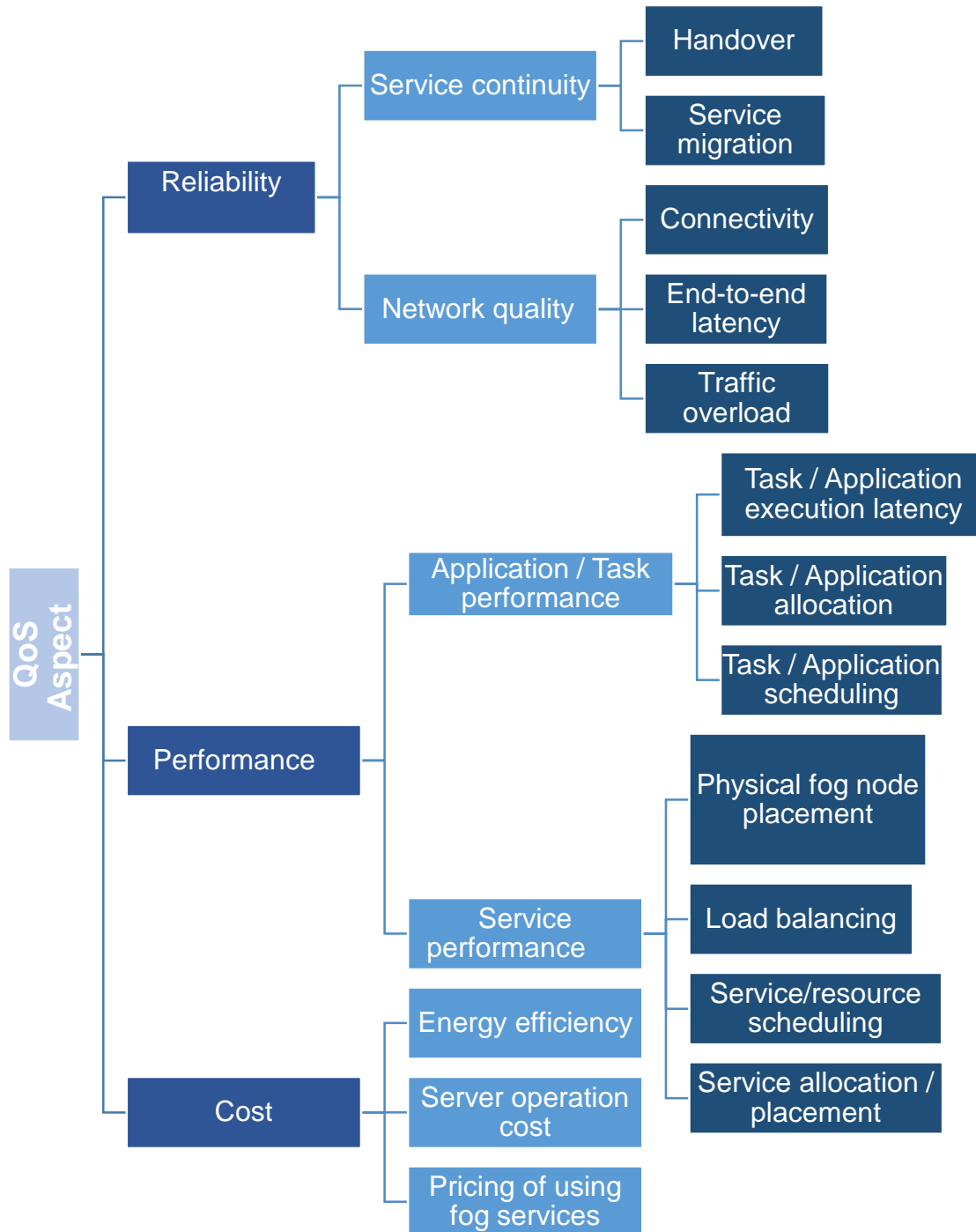


Figure 8: QoS aspect taxonomy tree

Reliability

Reliability of fog computing is centered on the two pillars that are service continuity and network quality.

Service Continuity

It should be noted that to achieve reliability in fog computing, service continuity plays a pivotal role. To achieve service continuity handover and service migration are at the center of it.

Handover

Handover is one such aspect which promotes service continuity and should be done with care. Handover in fog computing is the ability of the fog to continuously provide the services to the client whether they are stationary or moving from one point to another with continuous communication without service interruption (S. Wang et al., 2018). In order to continue providing the fog to the client, the service provider needs to figure out how the client can seamlessly access the machines that host the fog. The main objective behind the handover mechanism is to keep the connection between sensor nodes and a gateway with low latency (Gia et al., 2018). It becomes a greater challenge in handover when the size of the signal messages and state data to be transferred by fog terminals and fog stations becomes very large. This will cause an increase in the transmission overhead which is the cost of transmitting handover messages between two nodes. Moreover, processing overhead which is the cost of processing messages at each node in the network is also increased. In a nutshell, increasing the sum of handover signaling overhead delays the performance of handovers which result in compromising the reliability aspect of fog computing. Moreover, handover can also cause huge energy consumption (S. Wang et al., 2018). In order to address the above challenges of handover Gia et al., discussed and analyzed the metrics for handover mechanisms based on Wi-Fi (Gia et al., 2018). They proposed a handover mechanism that would support mobility in remote real-time streaming IoT systems. The mechanism promoted the connection of sensor nodes and the system whilst offering low latency. Authors in (Satria et al., 2017) suggested two Mobile Edge Computing (MEC) recovery schemes as to avoid degrading Quality of Experience (QoE) which is stated as a quality that is experienced by the users and is an extension of the QoS in the event that there is overload and or broken MEC. They applied optimal algorithms for allocation of ad-hoc relay nodes. Even though their simulation results demonstrated the mitigation of the problem of overloaded MEC, they did not look on how to improve performance by reducing the execution time if the system is implemented in different MEC environment in the event that mobile devices increase dramatically. To achieve low delay performance and service continuity in connected vehicles even when the resource is under heavy load (J. Li et al., 2017) proposed two resource management schemes, in fog enhanced radio access networks (FeRANs) which would prioritize real-time handover for moving vehicular services so that vehicular users can access the services with only one hop. Simulation results proved that their proposed solution would offer improved one-hop access achieving low delay performance.

Service migration/relocation

Service migration/relocation is also an important aspect of achieving service continuity. Service migration is similar to handover as it focuses on the mobility of users from one area to another. The difference is that in-service migration; the provider has to move the corresponding software

service from one machine to another machine in order to ensure the client to continue accessing the service while the client is moving. This implies that when transferring data, you need to consider time cost as it causes a greater challenge for seamless service migration. Another problem that can be experienced in-service migration is delays because of the existence of different network topologies between the start fog node and the destination fog server which has different transferring latency and process cost (S. Wang et al., 2018). Follow Me Edge (FME) concept was introduced by (Taleb et al., 2017) as to allow any time anywhere data access as to ensure high QoE and reduced latency. The research discovered that to achieve efficient migration of service, selecting the right combination of techniques is important. The researchers pinpointed out that for future work, there is a need to find out which combinations will achieve migration of services. In a bid to improve QoE in future wireless access networks, the authors (Iotti et al., 2017) used an internet access network approach based on fog computing where they dynamically moved content from cloud/web to nodes located at the edge of the access network. Their work played a pivotal role in optimizing bandwidth usage, reducing latency and enhancing QoE as validated by experimental analysis of the data collected from public Wi-Fi hotspots.

Network Quality

Network quality is another aspect which supports reliability and it can be achieved when we have good connectivity, low end-to-end latency, and less traffic overload.

Connectivity

Network connectivity can be defined as the average number of nodes connecting to other sets of nodes over a certain time duration under the link condition (Chuanmeizhi et al., 2016). This implies that when the connectivity of the network is so strong, there will be a decrease in the probability of reconnection which is as a result of the increase of the distance. In simpler terms, connectivity is the condition when a valid communication route between two or more nodes exists in order to exchange data packets (Artimy, Robertson, & Phillips, 2004). If there is connectivity failure due to several factors for example increase in distance between nodes, decrease in density of nodes, velocity in mobility situation, etc. will result in data being lost during those connectivity outage periods and cause some data to be delivered out of order. Due to these connectivity outage periods, severe delays can also be experienced which compromise time-critical application thus affecting QoS (Pérez et al., 2018). In the of vehicular networks, (Chuanmeizhi et al., 2016) concluded that the decrease of stability of connectivity leads to a smaller probability of reconnection of nodes especially when the communication range and average distance becomes larger. To solve this challenge, they propose that the density of nodes be increased in an area where there are many IoT devices as to avoid congesting the network thus creating strong connectivity between nodes which makes a network to behave as wired. In order to satisfy connection conditions in 5G systems, (Sampei, 2017) proposed that a controller functionality is shifted to an edge controller which will be located close to the controlled machine. To further enhance the flexibility of networking, network slicing and softwarization using the software-defined network (SDN) are the other two pillars which were suggested in this work to be included in the network. It was concluded that, it is best to make a flexible wireless access system to support the deviation of the number of simultaneously connected devices in enormous connections.

End-to-end latency

End-to-end latency which can also be called service delay is the response time (the time required to serve a request) that is the time interval between the moment when an IoT node sends a service request and when it receives the response for that request (Yousefpour, Ishigaki, Gour, & Jue, 2018) is another aspect of network quality which needs attention. One of the major causes of an end-to-end latency can be attributed to server compute time, network flexibility in terms of transfer speeds (Chen et al., 2017), a distance of the controller (server node) to the controlled machine (client node) (Sampei, 2017), etc. End-to-end latency is a challenge because of different wireless network communication systems coming into play (Wi-Fi, 3G, 4G, 5G, LTE-U) which does have different flexibilities. To reduce end-to-end latency and satisfy latency conditions in the new coming 5G, Sampei (2017) proposed that the location of controllers be near the controlled machine (Sampei, 2017). Moreover, Sampei (2017) further suggested that network functionality in wireless access ought to be made flexible as to fulfill end-to-end QoS requirement. Additionally, Souza et al. highlighted that another key point to note on controllers is, they should be a reduced number of involved control elements and average delay provided by each one of them if we are to enable low latency end-to-end communication (Souza et al., 2017). Authors Chen et al. provided some insights on how to reduce end-to-end latency by 60%-70% without sacrificing accuracy by introducing a novel black-box multi-algorithm which leverages temporal locality approach (Chen et al., 2017). The two parameters they used to measure their algorithm for an end-to-end latency was the processing and networking time (when the frame is captured to when the corresponding response is received). Researchers in (Yousefpour et al., 2018) introduced a general framework for IoT-fog-cloud applications and offloading policy for fog-capable devices that aims to reduce and minimize service delay for IoT applications. In their work, Chao Wang et al. introduced CPEP middleware for real-time cyber-physical event processing to reduce processing delay and enhance reduced end-to-end latency (Chao et al., 2017). The works of Aral and Brandic proposed a Bayesian Network model of QoS related parameters to predict the availability of virtual machine in edge infrastructure (Aral & Brandic, 2017). Their work wanted to limit deteriorating response time which is a critical factor in edge applications. Experimental results showed that the proposed method can identify virtual machines that satisfy user-defined availability objectives with up to 94% accuracy

Traffic Overload

A traffic overload challenge happens when a communications network exceeds the maximum finite volume of traffic it is supposed to carry. As a result, a degradation in performance will occur which will cause latency and some delays. It has become a challenge because of the IoT ecosystem which is generating a tremendous amount of data. As a result of that tremendous amount of data, it becomes difficult to transverse data over the network. This is a serious issue which affects network quality. A novel deep learning-based traffic flow prediction method was proposed by (Lv et al., 2015) which predicted the traffic flow and helped to prevent traffic overload which affected network and QoS. Fan and Ansari proposed a Load Balancing (LAB) scheme which allowed load balancing among the base stations in the event that the traffic load of the network is heavier than the computing load of the network (Fan & Ansari, 2018). Their findings demonstrated that LAB can perform better compared to the α -distributed algorithm and best Signal Interference Noise Ratio (SINR) algorithm.

The above work can be summarized in *Table 2* which highlights the QoS parameters being addressed in each paper

Table 2: Summary of QoS parameters addressed under reliability

	Service continuity		Network quality		
	Handover	Service migration	Connectivity	End-to-end latency	Traffic overload
(Satria et al., 2017)	TD				
(Gia et al., 2018)	TD; AB				
(J. Li et al., 2017)	TD; AB				
(Taleb et al., 2017)		TD; AB			
(Iotti et al., 2017)		AB; TP			
(Artimy et al., 2004)			AB; TD; TP		
(C. Wang et al., 2016)			TD		
(Sampei, 2017)			AB	TD	
(Pérez et al., 2018)			TP		
(Aral & Brandic, 2017)			AB,TD,TP		
(Souza et al., 2017)				TD; TP	
(Chen et al., 2017)				TD	
(Yousefpour et al., 2018)				TD	
(Lv et al., 2015)					PO
(Fan & Ansari, 2018)					PO

TD = Transit delay AB = Availability TP = Throughput PO = Priority

Performance

In order to have the best application and or task performance and service performance in fog computing, there is a need to handle the following in a cautious way.

Application/Task performance

The three pillars to achieve effective and efficient application/task performance are lowering task/application execution latency, improve task/application allocation and improve task/application scheduling.

Task/Application execution latency

Task execution latency is the total duration of transmission of a task from the IoT devices to the fog/edge servers, queuing and processing at the fog servers and its return with a successful reception at the IoT devices (Dao et al., 2018). The greater the task execution latency, the poorer the performance of that system. Reducing task execution latency is the key to achieve good performance. QoS and QoE are seriously affected by execution latency as it generally lowers the performance which affects task response time. Therefore, when you want to reduce task execution delay in the mobile edge network, it is wise to consider user mobility, task properties,

and network constraint. Wang et al proposed a light weight heuristic algorithm solution which provided accurate delay estimation (fast scheduling) to support accurate offloading decisions on mobile devices(Z. Wang et al., 2018). In their findings through simulation experiments to test for performance, it was noted that end-to-end delay for MEC can be reduced significantly through task execution delay which leads to an increase in resource utilization. Researchers in (Chowdhury et al., 2018) improved the task execution latency when they implemented the context-aware task migration scheme for HART-centric task execution in fiber-wireless (FiWi) based Tactile Internet infrastructures. Apart from selecting suitable cobot and collaborative node for HART-centric task execution, their approach would also migrate a task from one collaborative node to another. Based on their findings, the proposed task migration scheme proved without reasonable doubt that it is suited to provide low-latency performance for emerging Tactile Internet applications. A pattern-identified online scheduling task (PIOTS) mechanism was introduced by (Dao et al., 2018) to help assign task as a way to address task processing latency and service capabilities challenges as to satisfy industrial IoT applications. To achieve its goal, PIOTS scheme uses SOM technology to identify a task, then assigns the task to appropriate ECE by using Hungarian method. Thus deliberating on real-time task assignment.

Task/Application allocation

Task or application allocation is whereby a task or application is assigned to a fog server/node and the cloud server depending on the task requirement. Assigning a task to a server/node that meets its requirements helps in processing the task without delays. This helps in meeting user QoS requirements, reduces latency and facilitates quick response time. In such an environment where there are heterogeneous and autonomous devices, there is a need to have low complexity algorithms that help for efficient task allocation among nearby device. However, to come up with such low complexity algorithm is inherently a challenging problem which requires serious attention so as to fulfill user QoS requirements on task allocation (Dana Jošilo & Dán, 2018).

Task/Application scheduling

Task scheduling is the ability to schedule a task to fog nodes that will execute a task at the shortest time. This is achieved when a task is scheduled to a node with high computational power. The objective is to satisfy the user QoS requirements and to improve the fog computing throughput. Failing to select appropriate resources for the application task is referred to as task scheduling problem in fog environment. (L. Liu et al., 2018). Neil et al proposed a resource allocation strategy for fog computing using Priced Timed Petri Nets (PTPN) which helped to utilize and link both cloud and fog resources. Their approach helped to improve efficiency of resource utilization, satisfy user QoS requirements and maximize the profit of both providers and users which has become a big challenge. Priced Timed Petri Nets technologies allowed the user to choose the satisfying resources autonomously from a group of pre-allocated resources. Based on the results, the authors concluded that their approach can achieve more efficiency when compared to static allocation strategies based on task completion time and price. The work of (Xiao & Krunz, 2017) proposed a novel offload forwarding strategy where fog nodes would either not offload; offload and forward part or its entire load so that it will be processed by other local fog nodes which are idle and have better computational power than them. This strategy helped to minimize the average response time which included workload transmission time and queuing delay at the fog layer and significantly improved the performance of fog computing network, thus improving Quality of Experience (QoE) of users. The researchers

validated their approach using traditional Alternating Direction Method of Multipliers (ADMM) approach which proved that it cannot be used to solve the offload allocation problem for fog computing. However, their work used fewer nodes and did not pay attention to time-critical events which call for further experimental trials to check how it will perform in such a scenario.

Service Performance

Service means the fog service. Not the physical machine, neither the application that runs on the fog server. A fog service can be a Virtual Machine service or a Docker container service that allows the client to deploy application/task on it. Service performance can be achieved by physical fog node placement, load balancing, service/resource scheduling, and service allocation/placement.

Physical fog node placement

Where to place the physical equipment/machine that provides the fog is an important factor in-service performance. The greater the distance of fog nodes to the machines/ devices it wants to serve, the lesser the service performance and this will have a negative effect on QoS. If fog node placement problem is not handled carefully, it means service performance will be affected causing more task processing delays. The work of (Yao et al., 2017) acknowledges the importance of fog computing which pushes the network resources closer to the user in addressing QoS. The authors proposed that the cloudlet servers should be deployed on a given set of access points where users randomly roam among them with known statistics. However, they reasoned that fog computing cannot be used alone in supporting mobile computing task considering the fact that cloudlet servers are heterogeneous (have different resource capacities). To support the physical placement of access points and computational complexities, they further propose and devised a low-complexity heuristic greedy in principle algorithm with polynomial-time complexity and applied the Barabasi-Albert Model to generate random networks. Experimental results proved the efficiency of their algorithm as it addresses the heterogeneous problem of cloudlets in fulfilling predetermined QoS.

Load balancing

Load balancing goal is to distribute efficiently and fairly the dynamic workload across multiple nodes to ensure that no single node is overwhelmed. It is very important to do load balancing as it helps in optimal utilization of resources, reduces energy consumption, enabling scalability, avoiding bottleneck and over-provisioning and reducing response time. Additionally, in case of service fail, load balancing helps in continuation of the service by provisioning and de-provisioning of instances of the application without fail thus implementing fail-over (Jain Kansal & Chana, 2012). It can be noted that the above-highlighted advantages of load balancing enhance the performance of the system. According to (Neto, Callou, & Aires, 2017) load balancing comes with many issues related to QoS, security, and networking which are a cause of concern in fog computing and if not handled carefully, they negate the advantages of implementing fog computing. After the realization that fog computing face challenges such as multi-tenancy optimization and load balancing, (Neto et al., 2017) introduced a Multi-tenant Load Distribution Algorithm for Fog environments (MtLDF). This helped to optimize the load balancing in fogs environments considering specific multi-tenancy requirements (delay and priority). Authors in (X. He, Ren, Shi, & Fang, 2016) developed novel SDN-based modified constrained optimization particle swarm optimization (MPSO-CO) centralized load balancing algorithm which helped to balance workload between cloud/fog devices. This approach helped in reducing task processing latency challenges which affect latency-sensitive services on the Internet of

Vehicle (IoV). Their simulation results showed a decrease in latency and enhanced QoS which assisted in the latency-sensitive task. In their future work, they reiterated the need to do research in other load balancing algorithm and also look at other aspects of QoS such as security, capacity, etc.

Service/resource scheduling

Resource scheduling is one major issue of fog computing, the scheduling policy and algorithms affect the performance of fog computing directly. Resource scheduling is to provide an optimal mapping that assigns a required task or virtual resources onto available fog (or physical) resources at a specific time (Zhan et al., 2015). Resource scheduling should involve managing and scheduling fog resources and the taxonomy should consist of basically three categories that are *scheduling in the application layer*, *scheduling in the virtualization layer*, and *scheduling in the edge layer*. Most scheduling problems consist of four basic elements that are resources, task, objective (things that need to be fulfilled) and constraints. It is very important to address resource scheduling problem because there are cases where two tasks may have to share one resource. In doing resource scheduling in fog computing, the main objective will be to minimize service latency. However, it is noted that resource scheduling in fog computing is affected by other delay components such as transmission delay, queuing/networking delay, processing time, dependency constraints and resource queues which is a problem which requires serious attention as to minimize service latency in fog computing and promote improved QoS. Two-stage Stackelberg game approach and two computation offloading algorithm which assisted in offloading computation from cloud to local fog servers which are available at the edge was introduced by (Y. Liu et al., 2017). In their work, the fog servers would join the network and leave dynamically. This approach helped to offer low delay and reduced complexity thus providing satisfied QoS. Critically analyzing their base which they used to draw a conclusion, it can be noted that the authors validated their results using theoretical analysis which cannot be a standing measure. As such, there is a need to test this approach using other experimental means.

Service allocation/ placement

Service placement objective is to place each service either on a fog cell/ virtualized fog resource while taking into consideration factors like QoS guided by limitations like deadlines on the execution time of applications. It is important to allocate service to a resource that suits user QoS requirements to minimize execution delays which affect response time. The key principle guiding service placement in fog computing is to maximize the utilization of fog landscape and adhere to the QoS expectations of the application (Skarlat et al., 2017b). Failing to do service placement will affect execution time resulting in some delays which affect the QoS requirement. With the same motive of addressing the challenge of fog service placement, (Skarlat et al., 2016a) proposed a conceptual framework for fog resource leasing and releasing (provisioning). The envisioned architecture was evaluated using a customized simulation. It was observed that the approach decreased task request delays by 39%. In a bid to address some shortfalls highlighted above, authors (Skarlat et al, 2017b) implemented the system in iFogSim testbed as to solve the fog Service Placement Problem (FSPP) whilst considering the heterogeneity of applications and resources in terms of QoS attributes. They introduced a generic algorithm which assisted in reducing network communication delays and promoted a better utilization of fog resources. Simulation results showed an improvement in service placement plan produced by the genetic algorithm, greedy first-fit heuristic, and an exact optimization method. Being

motivated to investigate the optimal placement of cloudlets and with the motive to combat highly dynamic traffic loads of mobile IoT device which cause access delays (affecting QoS) and in addition addressing the challenge caused by heterogeneous infrastructure among IoT networks (Zhao et al., 2018) applied a ranking-based near-optical placement algorithm (RNOPA) which is an improved version of EOPA which was able to dynamically adapt to mobile IoT and their traffic loads. Their experimental and extensive simulation results showed improvement both in average cloudlet access delay and reliability when using RNOPA compared to Kmedians Clustering algorithm. However, in their work, they did not look at latency and stability when offloading tasks from the overloaded access point to a remote cloud.

The above work can be summarized in *Table 3* which highlights the QoS parameters being addressed in each paper

Table 3: Summary of QoS parameters addressed under performance

	Task Performance			Service Performance			
	Task Execution latency	Task allocation	Task Scheduling	Physical fog node placement	Load balancing	Resource Scheduling	Service Allocation
(C. Wang et al., 2016a,)	TD; PO						
(Z. Wang et al., 2018b)	TP; TD		AB				
(Chowdhury et al., 2018)	TD;TP	AB; PO					
(Dao et al., 2018)	TD		AB; PO; TD				
(Song et al., 2017)		TD; TP					
(Ni, Zhang, & Yu, 2018)		AB; TD; TP					
(Xiao & Krunz, 2017)		AB; TP					
(Rashidi & Sharifian, 2017)		AB;TD,TP					
(Shen et al., 2015)		AB; TD; TP					
(K. Wang & Yang, 2017)		TP					
(Alsaffar et al., 2017)		TD; TP					
(Pham et al., 2017)			TD; TP				
(G. Li et al., 2018)			AB; TD				
(Mahmud et al., 2016)			TD; TP; PR				
(Yao et al., 2017)				TD			
(Neto et al., 2017)					TD; PO		
(X. He et al., 2016)					TD; TP		
(Y. Liu et al., 2017)						AB;TD;TP	
(Skarlat et al., 2016)						AB; TD;TP	TP
(Taneja & Davy, 2017)							PO; TD
(Skarlat et al., 2017a)							PO; TD
(Skarlat et al., 2017b)							TP
(Zhao et al., 2018)							TD
(Y. He et al., 2018)							TD; TP
(Yang et al., 2018)		AV; PO					
(Kiani & Ansari, 2018)						TD,TP	

TD = Transit delay

AB = Availability

TP = Throughput

PO = Priority

Cost

The main reason for implementing fog computing to support cloud computing is to reduce the total cost because using cloud servers was now becoming expensive. To achieve that there is need to deal with energy efficiency, server operational cost, and pricing of using fog services.

Energy efficiency

Power efficiency is a measure of the amount of power spend by a node when doing specific tasks like processing, execution, offloading, etc. Because power utilization is a critical aspect of fog computing, it is vital that fog nodes maximize power efficiency by reducing the power consumed when processing the workload. It should be noted it is not only hardware efficiency that determines energy consumption, but factors such as resource management system that are deployed on the infrastructure play a role not forgetting the efficiency of the applications running in the systems. The total amount of power consumed by a node determines its effectiveness. the users' QoE and fog nodes' power efficiency are closely related to each other. The ability of a fog node to use less energy (power) even if it is given more workload to process plays a great role in achieving QoE as most IoT and fog nodes do not have bigger power reserves. Performance growth is also limited when there is higher energy consumption in fog computing systems as a result of carbon dioxide footprints and huge electricity bills. Therefore, it is a mandate to see to it that there is minimized power and energy consumption to minimize cost so as to improve the profits of using fog computing. End users are affected when there is no energy efficiency because more total cost will be incurred as a result of resource usage cost which is incurred by the resource provider (Beloglazov et al., 2011). The works of (Chowdhury et al., 2018) (Xiao & Krunz, 2017) (Shen et al., 2015) (K. Wang & Yang, 2017) implemented several strategies in addressing the energy efficiency challenge whilst addressing other QoS aspect as highlighted above. The works of (Kiani & Ansari, 2018) who introduced Non-Orthogonal Multiple Access (NOMA) optimized framework which is an edge-aware technique directly dealt with energy efficiency as its main objective as to reduce MEC users' uplink energy consumption. The NOMA minimized energy consumption by optimizing the user transmit powers, clustering and computing and communication resource allocation. Additionally, an efficient heuristic algorithm for user clustering was introduced for power control to be solved independently per NOMA cluster. Their results proved that the NOMA scheme can lower energy consumption.

Server operation cost

It is important to always minimize the server operational cost without compromising service performance. However, in a bid to reduce server operational cost by reducing e.g the number of fog layer nodes still faces challenges in dynamically supporting low latency services. Subsequently, the reduction of fog servers within the vicinity of the IoT devices to provide more computational resources in a bid to lower server operational cost will also lead to users' tasks accumulating resulting in the violation of the required service response time (Yang et al., 2018). It is still a problem to reduce server operational cost without compromising user QoS requirements. Researchers in (Yang et al., 2018) introduced a novel dynamic resource allocation framework in an effort to incur minimum operational cost whilst satisfying the applications' latency requirements. Their results ensure that the MEC service response was minimized

while achieving up to 33% operational cost reduction when compared to the fixed-location practices. Their approach did not consider the cost when there is a need to migrate from user-to MEC assignments

Pricing of using fog services

Customers who want to use computing resources from fog services are charged based on the number of virtual machines and hours of usage (Pham et al., 2017). It is also important to reduce the use cost if the fog has been in a long-term idle state or when it is assigned less task as to enable the user to be assigned to the resource that meets the requirement (Ni et al., 2018). The higher the pricing of using the fog service the less the task that is assigned to the resource and as a result the resource become long-term idle resulting in wasting resources.

The above work can be summarized in *Table 4* which highlights the QoS parameters being addressed in each paper

Table 4 Summary of QoS parameters addressed under cost

	Cost		
	Energy efficient	Server operation cost	Pricing of using fog services
(Shen et al., 2015)	AB; TD; TP;		
(Xiao & Krunz, 2017)	AB; TP		
(Alsaffar et al., 2017)	TD; TP		
(K. Wang & Yang, 2017)	TP		
(Chowdhury et al., 2018)	AB; PO; TD;TP;		
(Neto et al., 2017)		TD; PO	
(Pham et al., 2017)			TD; TP
(Ni et al., 2018)			AB; TD; TP

TD = Transit delay AB = Availability TP = Throughput PO = Priority

Open Challenges

As a recap and as supported by (Nath et al., 2018), QoS is achieved when there is reliability, low energy consumption, no or very minimum acceptable delay insensitive services, quality of experience for end-users and good network caching. Given these points, fog computing has played a pivotal role since its existence as evidenced in the above literature. Be that as it may, the ever increase in numerous dispersed devices which generate a huge amount of data that requires efficient processing opens another plethora and paused several open challenges. These challenges should be addressed through the designing of a deployable system over fog computing if we are to unceasingly benefit from fog computing advantages of offering in particular low latency, improved QoS among other benefits. Some of the open related challenges that affect QoS can be categorized as computing-related, management related, network and device-related challenges not to mention other general open challenges which include security-related challenges. In this section, our main focus will be to highlight open challenges that affect the QoS of fog computing as highlighted in the reviewed literature above.

Orchestration (Cloud-Fog) Challenge

It must be remembered that fog computing was not introduced to replace cloud computing but to assist in bringing some computational power to the ground. This will lead to transmission and processing of data to be done in close proximity with the devices. This assisted in addressing latency and QoS among other challenges faced when transmitting and processing data to the cloud. With this in mind, there is a need to continuously maintain and or developing effective and efficient orchestration mechanisms between cloud and fog. As highlighted in the literature, cloud-fog orchestration can cause other various open research challenges and if not handled carefully they can cause more delays hence affecting QoS.

Service or Tasks Partitioning

One such open research in this regard is to come up with an efficient and effective alternative for services or tasks partitioning between cloud and fog nodes. The solution should be centered on the idea of resources estimation, partitioning of task established on the availability of the resource at fog nodes in relation to the expected task completion and task response time. Moreover, optimal placement of sub-tasks should be done at various fog nodes and to the cloud. The works that have been done to address this challenge are application-specific and does not fully address fog computing challenges. As such, there is need to come up with a generalized solution that addresses the task partitioning problem in fog computing. The task migration solution to be developed should promote offloading from cloud to fog and fog to cloud with better response time and a high degree of accuracy without compromising QoS. Coming up with such a fog computing framework which supports the above-highlighted points especially requires deep and thorough research. Equally important, multi-domain orchestration should receive attention because failure to do so will become difficult to maintain resource allocation in multi-domain systems which are guided by heterogeneous policies. This is so because of the distributed nature of the fog nodes.

Enforcing Semantics in Fog Computing

In the context of the IoT environment, fog computing ecosystem is made up of several edge devices, actuators, heterogeneous sensors, cloud servers. In such a setup, it becomes a challenge to offer meaningful actions to perform the application. Without defining the action to be taken on an application, a device, sensor, actuator or servers can have a different service action to the same application which results in conflict and in-turn affect QoS. Therefore, enforcing semantics in fog computing by ensuring correct service in the workflow of the application is of paramount importance to achieve the end goal and avoid conflicting actions. Even though in literature there exist works that enforce semantics, unfortunately, they cannot be applied in fog computing due to the dynamic resource availability at each fog node and the role of clouds needs to be defined since fog is a partially distributed system. All these aspects need to be looked at as they affect QoS with respect to fog computing scenarios. Additionally, interaction among fog devices should be taken care of so that these interactions do not affect the overall system in generating the response in a short space of time as to avoid promoting response delays, especially in time-critical applications. This is because fog nodes always depend on one another when they are doing the task

Computing Challenge

Albeit, virtualization concept has played a pivotal role in fog computing in promoting effective resource allocation as to support orchestration of application services, there are several challenges that have arisen which need to be addressed so as to support virtualization architecture at the fog layer. Even though there are works which were done, most of them are not meeting the container resource allocation or VM characteristics for fog devices as highlighted in (Nath et al., 2018). Thus creating a research gap in container resource allocation or VM, which prompt for efforts of coming up with a solution that will properly execute some tasks in a manner that support time-synchronization. Furthermore, container migration or VM has opened other research gap as it should make the system fault-tolerant and fail-safe. Even though (Bittencourt et al., 2016) tried to address that, the proposed architecture did not cogitate service dependability as it initiates the migration which is an important factor to consider. Since fog computing is resource-constrained, virtualization should be lightweight. Because of that, several research challenges have risen which advocate for optimization frameworks that support the optimal placement, fog devices resource availability, the response time for a specific placement, container initialization delay and result aggregation

Management Challenges

As pinpointed in previous sections that fog devices are resource-constrained, it is of paramount importance that the resources are properly distributed or else QoS will not be met. This opens other research directions in this regard.

Resource Estimation and Allocation in Fog Nodes

Comparatively, to ensure fairness and QoS when doing resource allocation has become another research challenge in resource management at fog devices. Fairness in resource allocation will help in meeting the end-users QoS requirements. As such other application that requires real-time response has to be given high priority in assessing bandwidth as compared to those that are not time-critical. Even though there are some works done, in their future work they recommended the need to come up with a universal framework that is not domain-specific. It is a mammoth task to come up with a general framework that ensures fairness with service differentiation and QoS. As such, for fog based systems especially with the incoming of IoT becomes a challenging problem.

Management of Network Resources in Fog

Equally important is the management of network resources in fog especially in ensuring the correct network connectivity among the resources. Failure to provide an effective and efficient network resource management middleware will result in network congestion which in turn delay in response time hence affecting QoS. Even though Software Defined Networking (SDN) technologies have been implemented as to control resource in the fog based systems, there is still need for more work to be done since management of network resources is still a challenging task in a fog computing context.

CONCLUSION

With the dream of living in a “smart” world where IoT becomes the driving force of that world, many aspects in computing need many changes as to get a real-time response from every gadget as to provide quality of experience and quality of service for end-users. In such an ecosystem, minimizing response delays and providing a high quality of service plays a pivotal role. Fog computing technology has offered handshaking for the QoS for cloud, minimized latency thus supporting time-critical events and ushered better QoS for users. This survey explored works in the literature which implemented fog computing in existing systems to offer QoS, then further went on to look at works done with the intention of improving QoS in fog computing. All things considered and grounding our conclusion based on this current survey, it can be noted that fog computing can be the pillar in offering better QoS in geographical distributed IoT devices. Without any reasonable doubt, more work still needs to be done with the main intention of improving QoS in fog computing if we are to continuously benefit from fog computing as highlighted on the open challenges.

REFERENCES

- Albishi, S., Soh, B., Ullah, A., & Algarni, F. (2017). Challenges and Solutions for Applications and Technologies in the Internet of Things. In *Procedia Computer Science* (Vol. 124, Dec, pp. 608–614).
- Albishi, Saad, Soh, B., Ullah, A., & Algarni, F. (2017). Challenges and Solutions for Applications and Technologies in the Internet of Things. In *Procedia Computer Science* (Vol. 124, Dec, pp. 608–614).
- Alsaffar, A. A., Hung, P. P., Huh, E., & Korea, S. (2017). An Architecture of Thin Client-Edge Computing Collaboration for Data Distribution and Resource Allocation in Cloud, (November).
- Aral, A., & Brandic, I. (2017). Quality of Service Channelling for Latency Sensitive Edge Applications. *Proceedings - 2017 IEEE 1st International Conference on Edge Computing, EDGE 2017*, (11 September), 166–173.
<https://doi.org/10.1109/IEEE.EDGE.2017.30>
- Artimy, M. M., Robertson, W., & Phillips, W. J. (2004). Connectivity in Inter-Vehicle Ad Hoc Networks. *Ccece 2004, May*.
<https://doi.org/10.1109/CCECE.2004.1345014>
- Atlam, H. F., Walters, R. J., & Wills, G. B. (2018). Fog Computing and the Internet of Things: A Review. *Big Data and Cognitive Computing*, 10(April 2018).
- Beloglazov, A., Buyya, R., Lee, Y. C., & Zomaya, A. (2011). *A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems. Advances in Computers* (Vol. 82).
- Bittencourt, L. F., Lopes, M. M., Petri, I., & Rana, O. F. (2016). Towards Virtual Machine Migration in Fog Computing. *Proceedings - 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2015*, (03 March), 1–8.
- Bonomi, F., Milito, R., Zhu, J., & Addepalli, S. (2012). Fog Computing and Its Role in the Internet of Things. *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, (August 17, 2012), 13–16.
- Capossele, A. T., Cervo, V., Petrioli, C., & Spenza, D. (2016). Counteracting denial-of-sleep attacks in wake-up-radio-based sensing systems. In *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking, SECON 2016* (p. 03 November 2016).
- Chen, Z., Klatzky, R., Siewiorek, D., Satyanarayanan, M., Hu, W., Wang, J., ... Pillai, P. (2017). An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance. *Proceedings of the Second ACM/IEEE Symposium on Edge Computing - SEC '17, October*, 1–14.
- Chowdhury, M., ... E. S.-I. T. on, & 2018, U. (2018). Context-Aware Task Migration for HART-Centric Collaboration over FiWi Based Tactile Internet Infrastructures. *leeeexplore.lee.org*, 29, June(6), 1231–1246. Retrieved from
- Cisco. (2014). Quality of Service Overview. *Cisco IOS Quality of Service Solutions Configuration Guide*, (January 30), 18.
- Dana Jošilo, S., & Dán, G. (2018). Decentralized Algorithm for Randomized Task Allocation in Fog Computing Systems, 18 May.
- Dao, N.-N., Vu, D.-N., Lee, Y., Cho, S., Cho, C., & Kim, H. (2018). Pattern-Identified Online Task Scheduling in Multitier Edge Computing for Industrial IoT Services. *Mobile Information Systems*, 2018, 04 A.
- Dastjerdi, A. V., & Buyya, R. (2016). Fog Computing: Helping the Internet of Things Realize Its Potential. *Computer*, 49(8, August), 112–116.
- Ericsson. (2011). More Than 50 Billion Connected Devices. *White Paper*, (February),

1–12.

- Fan, Q., & Ansari, N. (2018). Towards Workload Balancing in Fog Computing Empowered IoT. *IEEE Transactions on Network Science and Engineering*.
- Gia, T. N., Rahmani, A. M., Westerlund, T., Liljeberg, P., & Tenhunen, H. (2018). Fog computing approach for mobility support in internet-of-things systems. *IEEE Access*, 6(June), 36064–36082.
- He, X., Ren, Z., Shi, C., & Fang, J. (2016). A novel load balancing strategy of software-defined cloud/fog networking in the Internet of Vehicles. *China Communications*, 13, 140–149.
- He, Y., Zhao, N., & Yin, H. (2018). Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach. *IEEE Transactions on Vehicular Technology*, 67-January(1), 44–55.
<https://doi.org/10.1109/TVT.2017.2760281>
- Iorga, M., Martin, M. J., & Feldman, L. (2018). Fog Computing Conceptual Model NIST Special Publication 500-325, (14 March).
- Iotti, N., Picone, M., Cirani, S., & Ferrari, G. (2017). Improving Quality of Experience in Future Wireless Access Networks through Fog Computing. *IEEE Internet Computing*, 21-March(2), 26–33.
- Jain Kansal, N., & Chana, I. (2012). Cloud Load Balancing Techniques : A Step Towards Green Computing. *IJCSI International Journal of Computer Science Issues*, Vol. 9, Ja(Issue 1), 238–246.
- Kiani, A., & Ansari, N. (2018). Edge Computing Aware NOMA for 5G Networks. *IEEE Internet of Things Journal*, 5-April(2), 1299–1306.
<https://doi.org/10.1109/JIOT.2018.2796542>
- Li, C., Xue, Y., Wang, J., Zhang, W., & Li, T. A. O. (2018). Edge-Oriented Computing Paradigms : A Survey on Architecture Design and System Management, 51(2, June 02 2018).
- Li, G., Song, J., Wu, J., & Wang, J. (2018). Method of Resource Estimation Based on QoS in Edge Computing, 2018(December 31).
- Li, J., Natalino, C., Van Dung, P., Wosinska, L., & Chen, J. (2017). Resource Management in Fog-Enhanced Radio Access Network to Support Real-Time Vehicular Services. In *Proceedings - 2017 IEEE 1st International Conference on Fog and Edge Computing, ICFEC 2017* (pp. 68–74).
- Liu, L., Qi, D., Zhou, N., & Wu, Y. (2018). A Task Scheduling Algorithm Based on Classification Mining in Fog Computing Environment. *Wireless Communications and Mobile Computing*, 2018august.
- Liu, Y., Xu, C., Zhan, Y., Liu, Z., Guan, J., & Zhang, H. (2017). Incentive mechanism for computation offloading using edge computing: A Stackelberg game approach. *Computer Networks*, 129(December 24), 399–409.
- Lv, Y., Duan, Y., Kang, W., Li, Z., & Wang, F. Y. (2015). Traffic Flow Prediction with Big Data: A Deep Learning Approach. *IEEE Transactions on Intelligent Transportation Systems*, 16(April 2015).
- Mahmud, M. R., Afrin, M., Razzaque, M. A., Hassan, M. M., Alelaiwi, A., & Alrubaian, M. (2016). Maximizing quality of experience through context-aware mobile application scheduling in cloudlet infrastructure. *Software - Practice and Experience*, 46(11, 29 December), 1525–1545.
- Nath, S. B., Gupta, H., Chakraborty, S., & Ghosh, S. K. (2018). A Survey of Fog Computing and Communication: Current Researches and Future Directions, (arXiv:1804.04365v1 [cs.NI] 12 Apr 2018), 1–47.
- Neto, E. C. P., Callou, G., & Aires, F. (2017). An algorithm to optimise the load

- distribution of fog environments. *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, (01 December), 1292–1297.
<https://doi.org/10.1109/SMC.2017.8122791>
- Ni, L., Zhang, J., & Yu, J. (2018). Priced timed petri nets based resource allocation strategy for fog computing. In *Proceedings - 2016 International Conference on Identification, Information and Knowledge in the Internet of Things, IIKI 2016* (Vol. 2018-Janua, pp. 39–44).
- OpenFog Consortium Architecture Working Group. (2017). 12 - OpenFog Reference Architecture for Fog Computing. *OpenFogConsortium*, (February), 1–162.
- Osanaiye, O., Chen, S., Yan, Z., Lu, R., Choo, K. K. R., & Dlodlo, M. (2017). From Cloud to Fog Computing: A Review and a Conceptual Live VM Migration Framework. *IEEE Access*, 5(11 April 2017), 8284–8300.
- Perera, C., Liu, C. H. I. H., Jayawardena, S., & Chen, M. (2014). A Survey on Internet of Things From Industrial Market Perspective. *IEEE Access*, 2(January 26,), 1660–1679.
- Perera, C., Zaslavsky, A., Christen, P., & Georgakopoulos, D. (2014). Context aware computing for the internet of things: A survey. *IEEE Communications Surveys and Tutorials*, 16(1, 03 May), 414–454.
- Pérez, J. L., Gutierrez-Torre, A., Berral, J. L., & Carrera, D. (2018). A resilient and distributed near real-time traffic forecasting application for Fog computing environments. *Future Generation Computer Systems*, 87, Octobe.
- Pham, X. Q., Man, N. D., Tri, N. D. T., Thai, N. Q., & Huh, E. N. (2017). A cost- and performance-effective approach for task scheduling based on collaboration between cloud and fog computing. *International Journal of Distributed Sensor Networks*, 13-11 Oct(11).
- Rashidi, S., & Sharifian, S. (2017). Cloudlet dynamic server selection policy for mobile task off-loading in mobile cloud computing using soft computing techniques. *Journal of Supercomputing*, 73(9, February), 3796–3820.
<https://doi.org/10.1007/s11227-017-1983-0>
- Sampei, S. (2017). Development of wireless access and flexible networking technologies for 5G cellular systems. *IEICE Transactions on Communications*, E100B(8, August 8), 1174–1180.
- Satria, D., Park, D., & Jo, M. (2017). Recovery for overloaded mobile edge computing. *Future Generation Computer Systems*, 70(14 July 2016), 138–147.
- Shen, H., Bai, G., Ma, D., Zhao, L., & Tang, Z. (2015). C2EM: cloud-assisted complex event monitoring in wireless multimedia sensor networks. *Eurasip Journal on Wireless Communications and Networking*, 2015(1, April).
- Simonet, A., Lebre, A., & Orgerie, A. C. (2016). Deploying distributed cloud infrastructures: Who and at what cost? In *Proceedings - 2016 IEEE International Conference on Cloud Engineering Workshops, IC2EW 2016* (pp. 178-183,04 August 2016).
- Skarlat, O., Nardelli, M., Schulte, S., Borkowski, M., & Leitner, P. (2017). Optimized IoT service placement in the fog. *Service Oriented Computing and Applications*, 11(4, October 04), 427–443.
- Skarlat, O., Nardelli, M., Schulte, S., & Dustdar, S. (2017). Towards QoS-Aware Fog Service Placement. *Proceedings - 2017 IEEE 1st International Conference on Fog and Edge Computing, ICFEC 2017*, (24 August), 89–96.
<https://doi.org/10.1109/ICFEC.2017.12>
- Skarlat, O., Schulte, S., Borkowski, M., & Leitner, P. (2016). Resource provisioning for IoT services in the fog. *Proceedings - 2016 IEEE 9th International*

- Conference on Service-Oriented Computing and Applications, SOCA 2016*, (26 December 2016), 32–39.
- Song, Y., Yau, S. S., Yu, R., Zhang, X., & Xue, G. (2017). An Approach to QoS-based Task Distribution in Edge Computing Networks for IoT Applications. *Proceedings - 2017 IEEE 1st International Conference on Edge Computing, EDGE 2017*, (11 September), 32–39.
- Souza, V. B., Gomez, A., Masip-Bruin, X., Marin-Tordera, E., & Garcia, J. (2017). Towards a Fog-to-Cloud control topology for QoS-aware end-to-end communication. In *2017 IEEE/ACM 25th International Symposium on Quality of Service, IWQoS 2017*.
- Taleb, T., Dutta, S., Ksentini, A., Iqbal, M., & Flinck, H. (2017). Mobile edge computing potential in making cities smarter. *IEEE Communications Magazine*, 55(3, March 13), 38–43.
- Taneja, M., & Davy, A. (2017). Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm. *Proceedings of the IM 2017 - 2017 IFIP/IEEE International Symposium on Integrated Network and Service Management*, (24 July), 1222–1228.
- Vaquero, L. M., & Rodero-Merino, L. (2014). Finding your Way in the Fog: Towards a Comprehensive Definition of Fog Computing. *ACM SIGCOMM Computer Communication Review*, 44(5, 10 October), 27–32.
- Wang, Chao, Gill, C., & Lu, C. (2017). Real-time middleware for cyber-physical event processing. *2017 IEEE/ACM 25th International Symposium on Quality of Service, IWQoS 2017*, (07 July).
- Wang, Chuanmeizhi, Li, Y., Jin, D., & Chen, S. (2016). On the Serviceability of Mobile Vehicular Cloudlets in a Large-Scale Urban Environment. *IEEE Transactions on Intelligent Transportation Systems*, 17, October(10), 2960–2970.
- Wang, K., & Yang, K. (2017). Power-minimization computing resource allocation in mobile cloud-radio access network. *Proceedings - 2016 16th IEEE International Conference on Computer and Information Technology, CIT 2016, 2016 6th International Symposium on Cloud and Service Computing, IEEE SC2 2016 and 2016 International Symposium on Security and Privacy in Social Netwo*, (13 March), 667–672.
- Wang, S., Xu, J., Zhang, N., & Liu, Y. (2018). A Survey on Service Migration in Mobile Edge Computing. *IEEE Access*, 6, 23511–23528.
- Wang, Z., Zhao, Z., Min, G., Huang, X., Ni, Q., & Wang, R. (2018). User mobility aware task assignment for Mobile Edge Computing. *Future Generation Computer Systems*, 85, March, 1–8.
- Xiao, Y., & Krunz, M. (2017). QoE and power efficiency tradeoff for fog computing networks with fog node cooperation. *Proceedings - IEEE INFOCOM*, (May).
- Xu, Y., & Helal, S. (2014). Application caching for cloud-sensor systems. In *Proceedings of the 17th ACM international conference on Modeling, analysis and simulation of wireless and mobile systems - MSWiM '14* (pp. 303–306, Septemeber 21).
- Yang, B., Chai, W. K., Xu, Z., Katsaros, K. V., & Pavlou, G. (2018). Cost-Efficient NFV-Enabled Mobile Edge-Cloud for Low Latency Mobile Applications. *IEEE Transactions on Network and Service Management*, 15(1), 475–488.
- Yao, H., Bai, C., Xiong, M., Zeng, D., & Fu, Z. (2017). Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing. In *Concurrency Computation* (Vol. 29).
- Yi, S., Li, C., & Li, Q. (2015). A Survey of Fog Computing: Concepts, Applications

- and Issues. *Proceedings of the 2015 Workshop on Mobile Big Data - Mobidata '15*, (15, June 21,), 37–42.
- Yousefpour, A., Ishigaki, G., Gour, R., & Jue, J. P. (2018). On Reducing IoT Service Delay via Fog Offloading. *IEEE Internet of Things Journal*, 5, April(2), 998–1010.
- Zhan, Z.-H., Liu, X.-F., Gong, Y.-J., Zhang, J., Chung, H. S.-H., & Li, Y. (2015). Cloud Computing Resource Scheduling and a Survey of Its Evolutionary Approaches. *ACM Computing Surveys*, 47, July(4), 1–33.
- Zhao, L., Sun, W., Shi, Y., & Liu, J. (2018). Optimal Placement of Cloudlets for Access Delay Minimization in SDN-based Internet of Things Networks. *IEEE Internet of Things Journal*, 5(2, April), 1–12.
- Zheng, X., Xu, L. Da, & Chai, S. (2017). QoS Recommendation in Cloud Services. *IEEE Access*, 5(19 April), 5171–5177.

A Fog Computing Framework for Quality of Service Optimisation in the Internet of Things (IoT) Ecosystem

William Tichaona Vambe
 Computer Science Department
 University of Fort Hare
 Alice, South Africa
 wvambe@ufh.ac.za

Prof Khulumani Sibanda
 Computer Science Department
 University of Fort Hare
 Alice, South Africa
 ksibanda@ufh.ac.za

Abstract—Fog computing plays a pivotal role in the Internet of Things (IoT) ecosystem because of its ability to support delay-sensitive tasks, bringing resources from cloud servers closer to the “ground” to support IoT devices that are resource-constrained. Although fog computing offers a lot of benefits such as quick response to requests, geo-distributed data processing and data processing in the proximity of the IoT devices, the exponential increase of IoT devices and large volumes of data being generated has led to a new set of challenges. One such challenge is the allocation of resources to IoT tasks to match their computational needs and QoS requirements whilst meeting task deadlines. Most proposed solutions in existing works suggest task offloading mechanisms where IoT devices would offload their tasks randomly to the fog layer. Of course, this helps in minimizing the communication delay, however, most tasks would end up missing their deadlines as many delays are experienced when fog node is deciding to process part of the task or offloading it to the next fog node. In this paper, we propose and introduce a Resource Allocation Scheduler (RAS) at the IoT-Fog gateway whose goal is to decide where and when a task is to be offloaded either to the fog layer or the cloud layer based on their priority needs, computational needs and QoS requirements and minimize round-trip time. The study followed the four phases of the top-down methodology. To test the efficiency and effectiveness of the RAS, a model was evaluated in a simulated smart home setup. The important metrics that were used are the queuing time, offloading time and throughput. The results showed that RAS helps in minimizing the round-trip time, increase throughput and improve QoS. Furthermore, the approach addressed the starvation problem, which was affecting low priority tasks. Most importantly, the results provide evidence that if resource allocation and assignment are done properly, round-trip time (queuing time and offloading time) can be reduced and QoS can be improved in fog computing.

Keywords—*Internet of Things, Fog Computing, Quality of Service, Resource Allocation*

INTRODUCTION

With the advent of the Internet of Things (IoT), which is creating a “smart world” and bringing about automation in many application areas, many computing elements need various modifications to support the IoT devices that are at the center of the automation world. Such modifications should support the IoT devices which are resource-constrained while keeping in mind that latency has to be minimized and Quality of Service (QoS) has to be improved.

As such, cloud computing was introduced to support IoT devices in terms of resources [1]. Although cloud computing concept dates back to the 1990s, this study found out the term

cloud computing was first used in 2006, precisely on the 9th of August by Eric Schmidt, Chairman and CEO of Google at the Search Engine Strategies Conference [2]. Since then, cloud computing has been widely adopted in many businesses for backup, file storage, cost-cutting in terms of infrastructure, development and testing as well as investment by cloud providers. Cloud computing has also been seen taking a central role to support the emerging IoT technologies in the interactions between IoT networks. However, the exponential growth of the number of connected sensors is becoming a challenge to cloud architecture. This is because cloud computing is a centralized approach which makes it not to be fully more appropriate to service geo-distributed IoT devices. The geographical distance seriously affects how the cloud servers and IoT devices communicate, leading to undesirable latency challenge. Secondly, it becomes costly to send the IoT generated tasks to and from the cloud servers as more bandwidth is needed during the transmission. This also negatively impacts on the latency of the requests.

Due to the above-mentioned challenges, fog computing was introduced by Cisco in 2012, not as a substitute for cloud computing, but to complement cloud computing [3]. Open Fog Consortium Architecture Working Group (2017), defined fog computing as “a system-level horizontal architecture that distributes resources and services of computing, storage, control and networking anywhere along the continuum from Cloud to Things” [4]. It is made up of both wired and wireless granular collection endpoints which include switching equipment, routers which act as gateways and customer premise equipment (CPE). Fog computing has become a preferred choice because of its ability to deliver services faster and also its ability to offer location awareness. It is worth to reiterate that fog computing technology is not a replacement of cloud computing but complements it by bringing the “cloud resources closer to the ground” where IoT devices reside [5]. As evidenced in the detailed survey done by [6], several studies have focused on various fog computing issues. One of the issues that is drawing much attention is how communication and computing resources can be allocated and assigned based on tasks requirements and priorities. The existing solutions, as informed in literature, indicate that resources are assigned/ offloaded based on a first come first serve basis without considering task status (whether a task is time-sensitive or not) and resource requirements [6]. Some existing works focus only on the reduction of communication delay. Despite many efforts being done to reduce communication delay, this study found

out that in many proposed solutions, most time-sensitive tasks fail to meet their deadlines. This situation can severely affect automation. The starvation problem is another challenge that is receiving much attention from various researchers. Another open challenge in fog computing is to find an effective and efficient resource allocation and assignment mechanism that meets the needs of both time-sensitive tasks and those that are not time-sensitive.

To address the above challenge and achieve this goal, this study proposed and implemented a Resource Allocation Scheduler (RAS) in the fog computing framework. RAS was introduced at the IoT-Fog gateways and its responsibility is to allocate and assign tasks generated by IoT devices to either fog layer or cloud layer based on the task's computational needs and priority.

The remainder of this paper is organized as follows: Section II is the related works; Section III is the description of the proposed framework. In Section IV, the RAS framework's component interactions are highlighted. Section V presents the evaluation setup. Section VI presents results and discussion. Section VII concludes the paper.

RELATED WORK

Literature shows that several researchers have used fog computing to minimise latency and improve QoS in existing systems. Such work includes the work of [7], who applied jitsi-meet and building process documentation cloud (BPDCC) applications to promote orchestration of services to address the QoS hindrance problem in the smart construction domain. The work of [8] discussed the issue of service placement in a home setup domain using a resource allocation algorithm to optimize data distribution and resource allocation. A two computational algorithm with low delay and reduced complexity that uses the principle of computation offloading in a mobile domain was used by [9] to address service migration mobility. A Follow me Edge (FME) concept was used by [10] in a smart city domain to achieve efficient resource deployment as a way to address the service migration problem. Modified Constrained Optimization particle swarm optimization (MPSO-CO) was applied by [11] on the Internet of Vehicles (IoV) domain to address load balancing challenges. Author [12] suggested combining network slicing, network softwarization, and MEC to address challenges faced when expanding cellular service to achieve efficient network flexibility. Power minimization resource algorithms MC-RAN was used by [13] in the mobile application domain to assist resource hungry and computational limited devices so that they will be able to dynamically compute resource allocation. The work of [14] devised a method of resource estimation. It was based on QoS in Edge computing, which used multi-attribute QoS resource matching algorithm and regression Markov prediction method to forecast available resources, select the suitable resource to meet the needs of users. Thus, reducing unnecessary competition for the resource, which improves QoS. The researchers in [15] argued that QoS is not only affected by data transmission factors but also processing delays in fog nodes. To address the end-to-end delay in fog computing, [15] introduced a service-oriented control that would allow control as a service (CaaS) in the fog to cloud topology. Fog Resource Reservation (FRR) and Fog Reallocation (FRL) strategies were introduced by [16] in fog

computing after the realization that fog nodes have limited resources when it comes to processing power. As such, they can quickly become overloaded when large amounts of users' requests arrive during peak hours, resulting in processing delays that will in-turn affect QoS. In [17], offload forwarding strategy was introduced to address service migration challenges in fog computing networks. A fog node would either not offload or offload and forward part or its entire load to be processed by other local fog nodes that are idle and have better computational power than it has. Task distribution algorithm, which was based on initialization, relaxation, rounding, and validation, was introduced by [18] to address the service migration problem in fog computing that affected QoS. [19] designed a novel Fog Service Placement Problem (FSPP) method that would facilitate optimal sharing of resources.

Although several related works tried to address QoS issues in IoT ecosystems, the challenge that remains is to be able to allocate and offload tasks to the resources that suit their computational needs and fulfil their QoS requirements. Moreover, to the resources that suit their deadline needs while minimizing roundtrip time [6]. Some researchers have made efforts to solve this challenge. For example, [20] and [21] have investigated and suggested ways on how to address the problem of task allocation and offloading. The latest research by [22] and [23] suggested offloading tasks to nearby fog nodes or cloud servers.

It is worth pointing out that all these works have one thing in common, the decision is made in the fog nodes to either process the whole tasks, part of the task or offload to the next fog node. This clearly shows that when tasks are sent to the fog layer, deadline requirements of tasks would not have been considered. Deadline requirements play a pivotal role when considering time-sensitive tasks as they require to be processed at a specific time frame. Failure to meet deadlines implies that if the outcome of the task comes after the stipulated time, it becomes useless. This can be detrimental in critical applications like medical health applications. Hence this study proposed a solution that would help tasks meet their deadlines by minimizing round-trip delays and addressing the starvation problem. The solution of this study introduced a Resource Allocation Scheduler (RAS) in the IoT-Fog gateways that is responsible for resource allocation giving high priority to time-sensitive tasks. The RAS considers task deadlines, resource constraints and promote minimized latency. This research is of paramount importance as several application areas such as smart health, smart city, smart grids would benefit from the findings of this research.

In the following section, we present our proposed framework and give a brief explanation of its components and interactions.

considered, that is (a) the arrival rate (ar) of the task and (b) the service rate (sr) of the computing device that is hosting the RAS. These two determine how the queue will move. Above

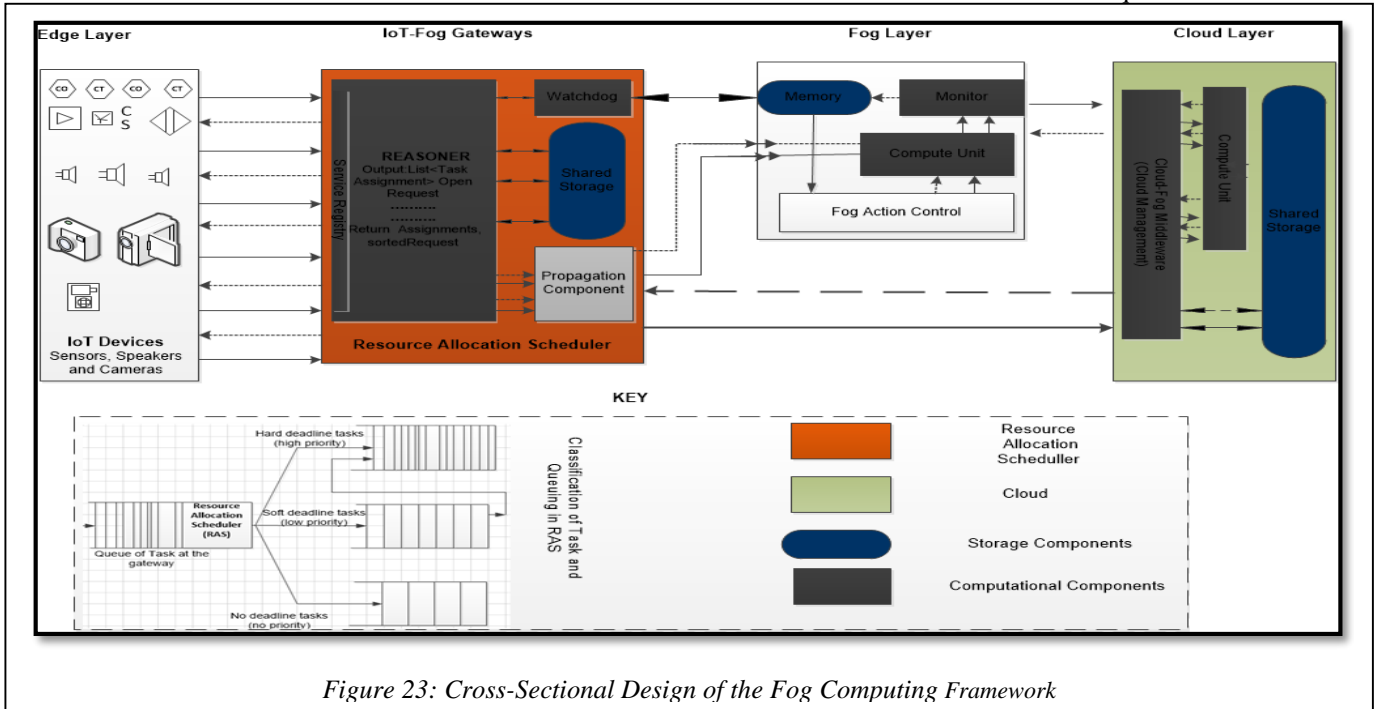


Figure 23: Cross-Sectional Design of the Fog Computing Framework

The framework is made up of the edge layer, IoT-Fog gateway where the Resource Allocation Scheduler (RAS) is hosted, fog layer and cloud layer as shown in **Figure 1**.

Edge Layer

In general, the edge layer comprises any IoT device that can connect using NFC [24], RFID [25], Bluetooth [26], Wireless Sensor Networks [27], Wi-Fi [27] and communicate together to perform some tasks or respond to events accordingly without explicit instructions. It should be reiterated that these devices have low computational power and storage capabilities.

IoT-Fog Gateways

IoT-Fog gateways are device in-between IoT devices and fog layer such as routers. It is in these gateways where RAS was introduced. The introduced RAS has a *service registry*, *reasoner*, *watchdog*, *propagation component* and *shared storage*, which are additional capabilities to address resource allocation challenges in fog computing.

When a task is sent from IoT to RAS, the *service registry* marks the task based on which IoT device it came from. It is the responsibility of the *reasoner* to do resource provisioning for the entire framework and make decisions to either send a task to the fog node or cloud.

The *reasoner* receives multiple numbers of tasks from different IoT devices that need to be assigned to either fog nodes or cloud servers. In the *reasoner* there will be a time-slotted system denoted by $ts = \{1, 2, 3, \dots, n\}$ and the time slot is denoted by AT . When there is no task to be assigned in the RAS, the queue denoted by Q will be empty, which means when $Q = \emptyset$ then $ts < 0$. The task will be arranged using the First-Come-First-Serve (FCFS)/ Q concept where Q represents the size of the queue. Using the Poisson process, it is considered that the time interval of arrival between successive task is exponentially distributed. There are two things to be

and beyond arrival rate and service rate, the moving of the queue is also affected by whether the computing devices in the fog layer or cloud layer are free or not at a certain time-stamp. The algorithm in the reasoner will classify the tasks into three main categories, namely time-sensitive task (high priority tasks), low time-sensitive (low priority tasks) and not time-sensitive (no priority tasks).

No-priority tasks are tasks that are not time-sensitive and do not have any stipulated time to be processed. Contrariwise, “high priority” tasks are time-sensitive and latency-sensitive tasks which should be processed within a specific time. If not processed, the task will no longer be valid for the IoT device. In almost a similar fashion with “high priority” tasks, “low priority” tasks are tasks whose processed output is valid up to a certain extent, if that time is not met, some penalties will be applied, but it will wait to be processed, and the IoT device will use that output even though the output will have failed to meet their corresponding deadlines.

These tasks would be placed in three different queues denoted by $Q = \{1, 2, 3\}$. Q_1 will be for high priority tasks, Q_2 for low priority tasks and Q_3 for no priority tasks. Q_1 and Q_2 tasks are sent to the fog node as they are time-sensitive whereas Q_3 tasks are sent directly to the cloud since they are not time-sensitive.

Q_1 tasks are given higher priority when compared to Q_2 tasks. Even though the tasks in Q_2 are not very time-sensitive when compared to Q_1 tasks, they should not suffer a starvation problem. The starvation problem occurs when Q_1 tasks keep on coming, which will result in Q_2 tasks not to be processed. Therefore, to avoid the starvation problem, after 10 seconds time-stamp, Q_2 tasks that are in the queue for a defined time without being assigned to any computing resource will be promoted by 1 to a higher priority queue.

Another point to note on the assignment of Q_1 tasks in the fog layer is that the tasks assigned to resources use a modified first-come-first-serve basis approach. That is, the first tasks to come is considered and assigned in the first available resource

if and only if the resource meets the tasks QoS requirements, computational requirements and user needs. If the first task requirements needs do not fit the available resource, the next task in the queue is considered and assigned to the available fog resource. This is done not to waste resources and waste the time of tasks in the queue that can fit and utilise the available resources. Thus, the modified first come first serve heuristic algorithm in the *reasoner* also helps in choosing the correct fog node for a specific task based on the QoS requirements of that task. Factors like distance, processing power of the fog nodes are considered as they play a pivotal role in time-critical tasks, as explained in previous sections.

The *watchdog* role is to monitor the status of the fog layer and cloud layer. If there is fault or errors at the fog node, events are triggered, and signals are sent to the reasoner. The reasoner then re-assigns the task that would have been processed in the fault node to the next available and capable fog node. These *watchdog* events are also put into considerations by the reasoner, as they help the reasoner to be more effective when making decisions.

The *propagation component* responsibility is to send the tasks to fog layer or cloud layer based on the decision which would have been made in the reasoner.

The purpose of *shared memory* in RAS is to hold a service registry and all the information of fog nodes and IoT devices that are registered in the network. Each fog node and IoT device are given unique identities, which would help to assign the correct response to the proper IoT devices.

Fog Layer

The fog layer is made up of any network resources, including mobile stations, servers, switches and routers depending on the area of application. These network resources offer their services to aid computation capabilities, pre-processing and temporary storage within the network and are named fog nodes [28]. Because of their proximity to the ground, they provide lower latencies compared to cloud computing which results in offering improved QoS. Fog layer receives Q1 and Q2 tasks from RAS and executes them before sending the response back to the RAS and to the cloud for the information that requires long-term storage.

Cloud Layer

LTE Communication Links are a middleware in the cloud. Their main function is to provide resource provisioning, task request placement and task execution in a specific cloud environment. The cloud is made up of servers whose resources are located in centralized data centers, whose responsibility is to process CPU intensive tasks and has bigger storage [29]. Since the cloud has virtually infinite resources, the tasks that require more computational power, more resources and are not time-sensitive, are sent by the RAS to be processed in the cloud. In this case, Q3 tasks are sent to the cloud layer

THE FRAMEWORK'S COMPONENT INTERACTIONS

Two things happen in this framework; new fog nodes can be registered, fog nodes can also be deregistered, and the task is processed.

Pairing and Service Deployment

When a new fog device wants to join the fog layer to give fog resources, it sends a signal to the RAS to be registered.

The RAS will register all the details, which include its processing power, RAM size, etc. of the new device, and it is instantiated as a fog node. Once the newly fog node is registered, there is no need to always take its details again because they will be stored.

Assuming that there is pending task requesting for a resource and the newly installed fog node does have the required QoS matrices requirements and resources for the task, RAS will immediately deploy the task. Once the service is deployed, the newly added fog node will be able to read and execute the task and return the response to the RAS. The same happens when a new IoT device such as sensors, actuators, laptops, smart television joins the network. It is registered at the RAS as a new IoT device and the type of data it sends is also recorded. This is done so that the RAS will keep that information to avoid repeating the process of identifying the type of data sent by the device each time it sends the data thus minimizing future delays.

Resource Allocation and Scheduling of Tasks

When a new task is sent from the IoT device, it goes through the RAS where it is labelled whether it is of high priority or not, specifying its QoS matrices requirements. If the task request is time-sensitive it is sent to the fog layer and if it is not time-sensitive, it is sent to the cloud layer for processing. If the decision by the RAS is to send the task request to the fog layer, the RAS will choose the most fitting fog node to deploy the task request. After the above reasoning is done by the RAS, it then deploys the task to the fog node. If the service is successfully deployed, the fog node sends a signal back to the RAS for evaluation purposes and the fog node starts immediately executing the intended functionalities on the deployed task. After the task has been executed in the fog node, the response is sent back to the RAS, which further forwards it to the specific IoT device. A copy of the response and other processed details are sent to the cloud for long term storage. If the task is not time-sensitive and requires more computational power, the RAS will flag it as such and deploy it to the cloud. In the cloud, it is assigned to the virtual machines (VMs) which process the request. The component, which will handle the task, will send a signal to the RAS as an indication that it was deployed successfully and for evaluation purposes too.

EVALUATION SETUP

To assess the proposed model, it is of paramount importance to define an assessment configuration with define limits, metrics and devices included. The assessment configuration in this project is based on an OSI model network topology. The network topology is made up of the edge layer (IoT devices), IoT-Fog gateways (hosting the RAS), fog layer (for processing time-sensitive task) and the cloud layer (for processing computer-intensive task).

In this network topology, each device should have a device name, the list of service types the device can send and or process, IP address and port, device location and location range. This information is important to avoid conflicts which might cause delays. Furthermore, it helps in identifying the location of the devices, whether they are at the edge, fog or cloud layer.

Simulation Description

In this research, the simulation was hosted on a high-performance computer with 1100 terabyte (TB) storage capacity, 135 cluster nodes with 2900 processor cores and 11TB memory. Twenty (20) IoT devices were used and the input data size was from 10MB to 30MB, while the output data size was 1MB to 30MB. Both input and output data sizes were uniformly distributed. Twenty (20) Mbps was used as a maximum transmission bandwidth. To evaluate and validate the results, 1000 independent runs were done and averaged for each parameter to get a better result output for the runs. Table 1 below presents the simulation parameters.

SIMULATION PARAMETERS

Parameters	Value
Number of IoT device	20
Number of IoT-Fog Gateways	3
Number of fog nodes	10
Number of cloud data centers	2
Number of tasks	100
IoT device CPU frequency	600x10 ⁶ cycles per second
IoT device memory capacity	128 Megabytes
Fog nodes CPU frequency	5 x 10 ⁹ cycles per second
Fog nodes memory capacity	512 Megabytes
Cloud server's CPU frequency	10 x 10 ⁹ cycles per second
Cloud server's memory capacity	64 Gigabytes
Maximum Bandwidth	20Mega Hertz

RESULTS AND DISCUSSIONS

Since our proposed solution was to reduce overall round-trip time, this study evaluated queuing time and offloading time which are round-trip time factors.

Queuing Time

Queuing time is the time a task waits in the queue before it is assigned to a fog or cloud resource. Queuing time plays a pivotal role in determining whether a task will be processed early or not based on how long a task would wait before it is assigned to a resource. The higher the queuing time, the higher the chances of an increased round-trip time, which has a negative impact on latency and affects QoS. Consequently, queuing time should be minimized to reduce round-trip time which is the motive behind this research. **Figure 2** presents the queuing time of tasks.

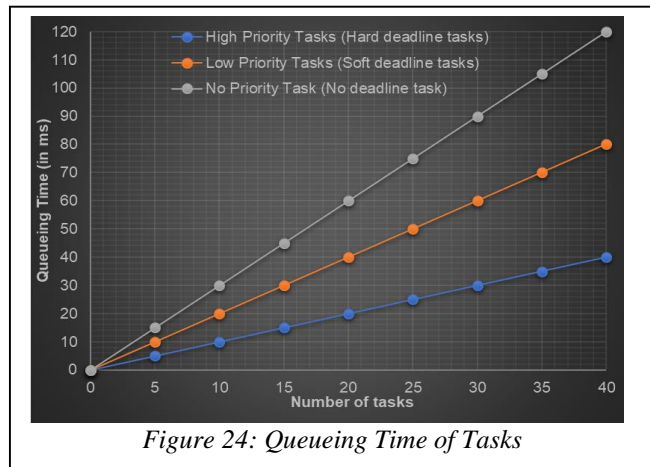


Figure 24: Queuing Time of Tasks

From **Figure 2**, it can be noted that for high priority-based tasks (blue line), the queuing times are minimal when compared to low priority-based tasks (orange line) and no

priority-based tasks (grey). Similarly, low priority-based queuing time is also minimal when compared to no priority-based tasks.

This is because high priority tasks are given preference during the assignment to both message routing and to the fog layer resources to be processed first as compared to the later. For this reason, high priority tasks are assigned and processed earlier than the other two, which gives them less queuing time. Correspondingly, the low priority tasks are given a better priority compared to those with no priority. No priority tasks take more time as they require more time in uploading.

It is also important to note that queuing time for both high priority tasks and low priority tasks can be further minimized if there are more fog nodes at the fog layer. The more the fog nodes at the fog layer, the less the queuing time experienced for high priority tasks and low priority tasks as there will be more options and resources to assign the tasks. Therefore, queuing time is directly affected by the number of fog nodes available at the fog layer.

Offloading Time

Offloading time is another factor that affects round-trip time. Offloading time is the time taken to upload, process and download a task from IoT device to RAS then either fog node or cloud depending on the task status and back to the IoT device. The more the offloading time, the greater the overall round-trip time. Moreover, offloading time is directly affected by queuing time. If the queuing time is minimized, the overall offloading time is also reduced. **Figure 3** presents the simulation result of offloading time.

From the graph, it is noted that offloading starts

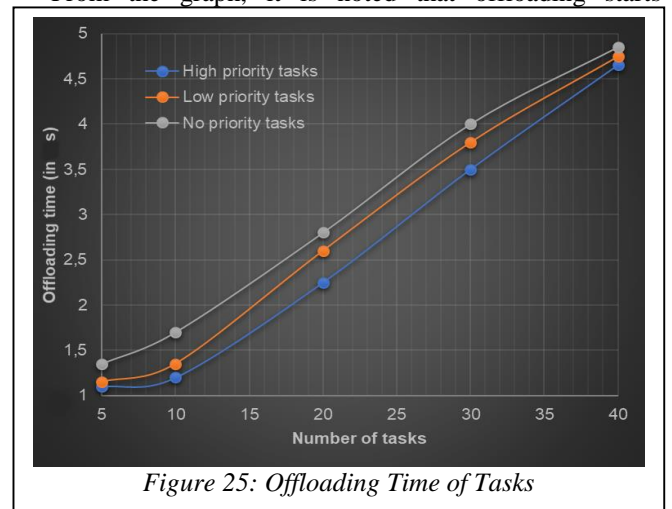


Figure 25: Offloading Time of Tasks

happening after 1ms. This is because some delays are experienced when tasks are generated and sent. The offloading time of all the tasks will increase as the number of tasks increases. This is because fog layer cannot handle many tasks at once especially when IoT devices generate a greater number of tasks that are time-sensitive and require fog layer resources. Another factor is, the more the traffic that wants to traverse the internet the higher the demand for bandwidth and also the higher the demand to resources which affect queuing time and offloading. Generally, high priority tasks have lower offloading time when compared to the other two because they are given first preference to resources and usually they are small in size. Low priority tasks have also lower offloading time when compared to no priority tasks.

Even though the no priority tasks were not being offloaded to the fog layer, they took more time to be offloaded to the cloud. The reason behind this is because of their size which requires more time to offload, process and download. The bigger the task, the more the time it took to be processed and traversed over the network. Moreover, it consumes more bandwidth.

Comparison of Processing in the IoT Device, Fixed Offloading to Fog Node and the use of RAS

In any scientific research, to be able to quantify how effective a proposed solution is, there is always a need to do a comparison of the proposed strategy with other existing strategies.

For comparison, we compared the average delay per user against the packet arrival rate of the fixed strategy method, RAS strategy and entirely processing the tasks on the IoT device strategy. A fixed strategy is when tasks are sent directly to the fog nodes from IoT devices, and the fog nodes would decide to either process the whole task, part of it or to send to the cloud. This strategy is the one being used by many researchers in literature.

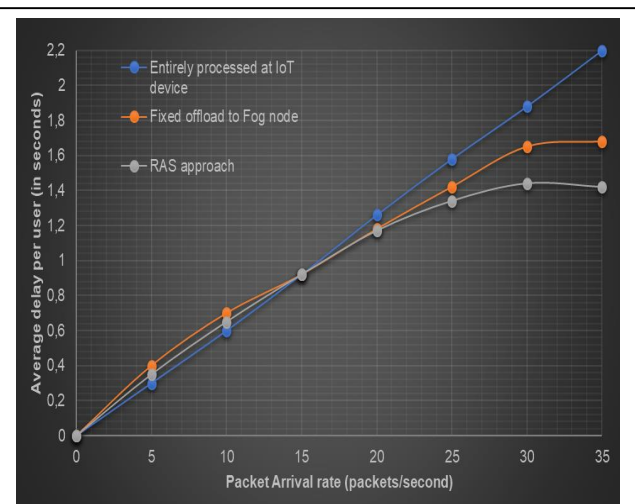


Figure 26: Task arrival, task execution and task offloading of different approaches

In **Figure 4**, at a packet arrival rate of between 0 and 14, the average delay per user is less when the task is entirely processed in the IoT device itself than when it is sent to the RAS or fixed offloaded to the fog node. Wholly, if there are few tasks that are time-sensitive and are not CPU intensive, processing them in the IoT device is much better than transferring them to the fog node, as this increases the average delay per user.

It can be noted in **Figure 4** that at 15 packets/sec arrival rate the average delay per user is the same for all the three strategies. This convergence point can be referred to as a point of equilibrium. At 15 packets arrival rate, the summation of the delays and the resources available at both the fixed offload to fog node approach and the RAS approach will be equal to the one processed entirely at the IoT device. At this point, it means using either of the approaches will give the same output.

After 15 packets/sec arrival rate, using other fog layer resources will be of greater benefit than to process the task at the IoT device as can be noted in the diagram that after 15

packets/sec arrival rate, the average delay per user of those tasks that are processed at IoT entirely keep on increasing when compared to the other two options. An increase in task production and packet arrival rate has a greater effect when the task is processed entirely at the IoT device level. The more the packets in the queue to be processed, the more the time needed to process them especially when they are processed in the IoT device itself. The packet arrival rate is directly proportional to the average delay per user if the task is processed in the IoT device. The demand for computing resources by the tasks can even cause the IoT device to end up being slow and not working properly. This is due to overloading as the IoT devices do not have much computational power, as such it cannot handle more tasks at once. Overloading IoT devices have a negative impact on the IoT device battery lifespan, as the device will be strained which results in using more battery power.

Again, referring to figure 4, it can be noted that between 15 to 20 packets per second arrival rate the performance of the fixed offloading and the RAS approach are almost the same. The difference can only be noted after 20 packets per second arrival rate when the RAS approach becomes better than the fixed offloading. The reason might be that the RAS approach chooses the best fog node for time-critical tasks when compared to the fixed offloading approach.

When tasks are using the RAS approach, it can be noticed that at first, using the RAS approach will only be better if compared to fixed offloading but worst when compared to those that are processed at the IoT devices. The delay experienced in time is because some time is used when transferring a task from the IoT device to the RAS before the actual processing of the tasks starts. The difference is noticed when there is an increase in the packet arrival rate where it can be observed that the RAS outclass both approaches highlighted earlier that are the one that allows tasks to be processed locally and that which uses a fixed offload approach. The reason being that the RAS chooses appropriate fog node to process the tasks when compared to the fixed offload approach, which sometimes offloads tasks to a fog node that does not satisfy the requirements of the tasks. As such, choosing the correct fog node that suits the requirement of the task first before assigning those tasks helps in the sense that when tasks are then finally deployed, it is guaranteed that they will be processed.

As evidenced and shown on the above figure and analysis, it can be deduced that when tasks are few, it is wise to process them in the IoT device. When there is a need to process more tasks, then processing them in the IoT device will no longer have benefits. The use of the RAS approach will be of greater advantage. We can conclude that the introduction of the RAS in the gateways, which makes decisions and give high priority to high priority tasks, proves to have more benefits compared to the two approaches mentioned above. The RAS approach minimizes round-trip time, improves throughput and as such, it improves QoS as compared to the later approaches. It is also important to note that the roundtrip time and throughput can be further minimized when more fog nodes join the fog layer, as the RAS will be able to deploy more tasks to different fog nodes resulting in the reduction of the queueing time for tasks.

RAS Strategy versus Other Resource Allocation Strategies

For comparison purposes, three strategies proposed and used in literature are considered and compared with RAS:

- **Strategy 1:** IoT devices would randomly choose a computing device either in the fog layer or cloud layer. Let us denote this scenario as *S1*.
- **Strategy 2:** IoT devices would choose a computing device with minimum uploading time. Let us denote this scenario as *S2*.
- **Strategy 3:** IoT devices would choose a computing device with sufficient CPU frequency for processing the tasks. Let us denote this scenario as *S3*.
- **Strategy 4:** Using the proposed Resource Allocation Scheduler strategy denoted with RAS.
 - *Performance-based on average queuing time*

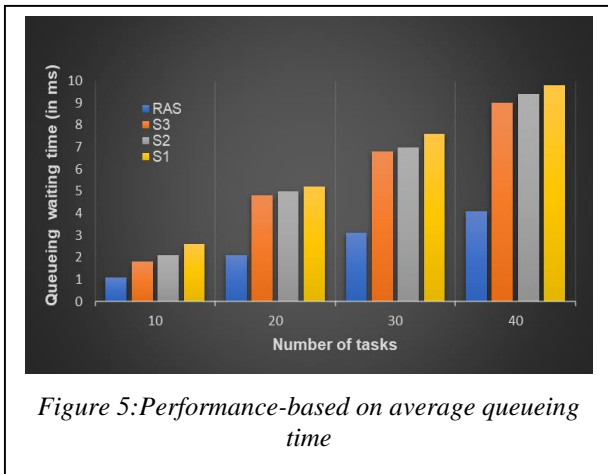


Figure 5: Performance-based on average queuing time

Based on **Figure 5**, which compares performance based on queuing time, our RAS improved performance when compared to the other three strategies as far as average queuing time is concerned. The average queuing time of high priority and low priority was minimized. This was as a result of them being given more priority if compared to those tasks with no deadline. It can be noted from **Figure 5** that even in the case of more tasks, the average queuing time of the RAS is less when compared to the other three strategies. These results proved that even if you are using first come first serve basis in different strategies if high priority tasks are not given high priority, that will affect the queuing time and has a negative impact on the time-sensitive tasks as QoS is compromised.

- *Performance-based on average offloading time*

Even if *S2* strategy allowed IoT devices to choose a computing device with minimum uploading time and *S3* allowed IoT devices to select a computing device with sufficient CPU frequency for processing the tasks, it could be observed from **Figure 6** that these strategies did not minimize offloading time as expected by IoT devices. Contrary to *S1*, *S2*, and *S3*, considering performance based on the average offloading time as shown in **Figure 6**, RAS managed to deploy tasks to computing devices that met the requirements of the task. Moreover, RAS offered a minimum communication overhead, which minimized round-trip time since offloading time was reduced when compared to other *S1*, *S2*, and *S3* strategies.

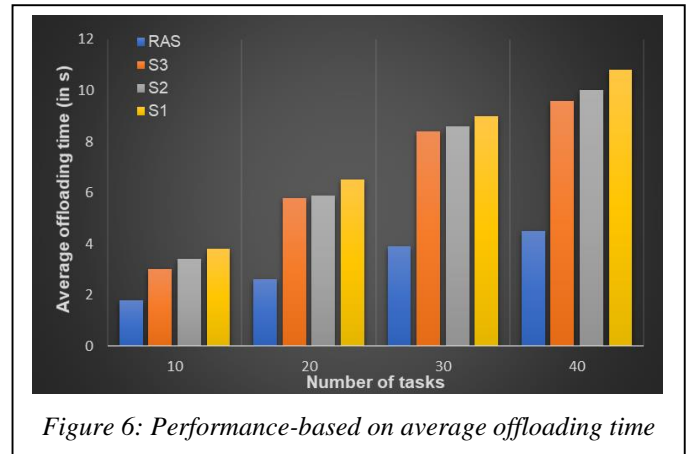


Figure 6: Performance-based on average offloading time

This was attributed to the fact that RAS would choose either fog node or cloud that satisfies the requirements of the task based on the task's status. Basing our argument on the simulation results, if the round-trip time is minimized, latency will also be reduced, and this will lead to improved QoS and improved performance.

- *Performance-based on average throughput*

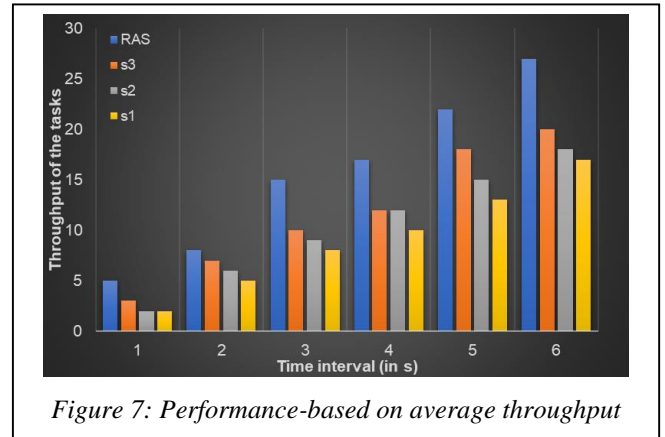


Figure 7: Performance-based on average throughput

To check if the RAS improved the QoS, throughput, which is one of the QoS parameters, was tested. In this work, throughput was calculated as the number of tasks that complete their process within a time-stamp based on the arrival rate. As indicated in **Figure 7**, RAS had high throughput when compared to other strategies. This is because different strategies failed to process more tasks within a given time-stamp. The RAS strategy managed to achieve improved throughput because it was able to deploy time-sensitive tasks to fog devices that met the resource requirements with minimum offloading time, which was also a factor of queuing time.

CONCLUSIONS

The results above show that even though many factors play a pivotal role in determining the total round-trip time, queuing time and offloading time are very important too. Minimizing the two will help in reducing round-trip time and subsequently leading to the minimization of latency. Also, if queuing time and offloading time are minimized, the overall throughput of the framework is significantly improved.

Another important finding from these results is that choosing a computing device with sufficient CPU frequency

for processing the tasks without considering other factors such as the type of the tasks, will not minimize average waiting time and average offloading time when more tasks are added. This kind of approach works well when there are small numbers of tasks that need to be processed. However, it suffers when many tasks need to be processed

ACKNOWLEDGEMENT

I would like to acknowledge the Govan Mbeki Research and Development Centre (GMRDC) and Centre of Excellence (CoE) for funding this research.

REFERENCES

- [1] Z. Chen *et al.*, "An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance," *Proc. Second ACM/IEEE Symp. Edge Comput. - SEC '17*, vol. October, pp. 1–14, 2017.
- [2] Google Press, "Conversation with Eric Schmidt hosted by Danny Sullivan," 2006.
- [3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," *Proc. first Ed. MCC Work. Mob. cloud Comput.*, no. August 17, 2012, pp. 13–16, 2012.
- [4] OpenFog Consortium Architecture Working Group, "12 - OpenFog Reference Architecture for Fog Computing," *OpenFogConsortium*, no. February, pp. 1–162, 2017.
- [5] C. Chang, S. N. Srirama, and R. Buyya, "Internet of Things (IoT) and New Computing Paradigms," in *Fog and Edge Computing: Principles and Paradigms*, 2017, pp. 1–23.
- [6] W. T. Vambe, C. Chang, and K. Sibanda, "A Review of Quality of Service in Fog Computing for the Internet of Things," vol. 3, no. 1, pp. 22–40, 2020.
- [7] P. Kochovski and V. Stankovski, "Supporting smart construction with dependable edge computing infrastructures and applications," *Autom. Constr.*, vol. 85, no. May 2017, pp. 182–192, 2018.
- [8] A. A. Alsaffar, P. P. Hung, E. Huh, and S. Korea, "An Architecture of Thin Client-Edge Computing Collaboration for Data Distribution and Resource Allocation in Cloud," no. November 2017.
- [9] Y. Liu, C. Xu, Y. Zhan, Z. Liu, J. Guan, and H. Zhang, "Incentive mechanism for computation offloading using edge computing: A Stackelberg game approach," *Comput. Networks*, vol. 129, no. December 24, pp. 399–409, 2017.
- [10] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck, "Mobile edge computing potential in making cities smarter," *IEEE Commun. Mag.*, vol. 55, no. 3, March 13, pp. 38–43, 2017.
- [11] X. He, Z. Ren, C. Shi, and J. Fang, "A novel load balancing strategy of software-defined cloud/fog networking on the Internet of Vehicles," *China Commun.*, vol. 13, pp. 140–149, 2016.
- [12] S. Sampei, "Development of wireless access and flexible networking technologies for 5G cellular systems," *IEICE Trans. Commun.*, vol. E100B, no. 8, August 8, pp. 1174–1180, 2017.
- [13] K. Wang and K. Yang, "Power-minimization computing resource allocation in mobile cloud-radio access network," *Proc. - 2016 16th IEEE Int. Conf. Comput. Inf. Technol. CIT 2016, 2016 6th Int. Symp. Cloud Serv. Comput. IEEE SC2 2016 2016 Int. Symp. Secur. Priv. Soc. Netwo.*, no. 13 March, pp. 667–672, 2017.
- [14] G. Li, J. Song, J. Wu, and J. Wang, "Method of Resource Estimation Based on QoS in Edge Computing," vol. 2018, no. December 31, 2018.
- [15] V. B. Souza, A. Gomez, X. Masip-Bruin, E. Marin-Tordera, and J. Garcia, "Towards a Fog-to-Cloud control topology for QoS-aware end-to-end communication," in *2017 IEEE/ACM 25th International Symposium on Quality of Service, IWQoS 2017*, 2017, no. 07 July.
- [16] J. Li, C. Natalino, P. Van Dung, L. Wosinska, and J. Chen, "Resource Management in Fog-Enhanced Radio Access Network to Support Real-Time Vehicular Services," in *Proceedings - 2017 IEEE 1st International Conference on Fog and Edge Computing, IC FEC 2017*, 2017, no. 24 August, pp. 68–74.
- [17] Y. Xiao and M. Krunz, "QoE and power efficiency tradeoff for fog computing networks with fog node cooperation," *Proc. - IEEE INFOCOM*, no. May 2017.
- [18] Y. Song, S. S. Yau, R. Yu, X. Zhang, and G. Xue, "An Approach to QoS-based Task Distribution in Edge Computing Networks for IoT Applications," *Proc. - 2017 IEEE 1st Int. Conf. Edge Comput. EDGE 2017*, no. 11 September, pp. 32–39, 2017.
- [19] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards QoS-Aware Fog Service Placement," *Proc. - 2017 IEEE 1st Int. Conf. Fog Edge Comput. IC FEC 2017*, no. 24 August, pp. 89–96, 2017.
- [20] S. W. Ko, K. Huang, S. L. Kim, and H. Chae, "Live Prefetching for Mobile Computation Offloading," *IEEE Trans. Wirel. Commun.*, 2017.
- [21] M. Mukherjee, S. Kumar, M. Shojafar, Q. Zhang, and C. X. Mavroumoustakis, "Joint Task Offloading and Resource Allocation for Delay-Sensitive Fog Networks," in *IEEE International Conference on Communications*, 2019.
- [22] Y. Yang, Z. Liu, X. Yang, K. Wang, X. Hong, and X. Ge, "POMT: Paired Offloading of Multiple Tasks in Heterogeneous Fog Networks," *IEEE Internet Things J.*, 2019.
- [23] Y. Wang, X. Tao, X. Zhang, P. Zhang, and Y. T. Hou, "Cooperative Task Offloading in Three-Tier Mobile Computing Networks: An ADMM Framework," *IEEE Trans. Veh. Technol.*, 2019.
- [24] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: Architecture, applications, and approaches," *Wirel. Commun. Mob. Comput.*, 2013.
- [25] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Networks*, 2010.
- [26] I. J. Dilworth, "Bluetooth," in *The Cable and Telecommunications Professionals' Reference: PSTN, IP and Cellular Networks, and Mathematical Techniques*, 2012.
- [27] W. Sensor, *WIRELESS SENSOR NETWORKS A Networking*. 2009.
- [28] A. A. T. R. Coutinho, F. Greve, and C. Prazeres, "An Architecture for Fog Computing Emulation," *Wcga - Sbrc*, vol. 15, no. 1/2017, 2017.
- [29] The National Institute of Standards and Technology, "The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology," *NIST Spec. Publ.*, 2011.

