UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

DESIREE DOS SANTOS

MiddleFog: A middleware to mitigate the latency in IoT communication devices

Thesis presented in partial fulfillment
of the requirements for the degree of
Master of Computer Science

Advisor: Prof. Dr. Cláudio Fernando Resin Geyer
Coadvisor: Prof. Dr. Julio César Santos dos Anjos

Porto Alegre
2024

CIP — CATALOGING-IN-PUBLICATION

Santos, Desiree dos

MiddleFog: A middleware to mitigate the latency in IoT communication devices / Desiree dos Santos. – Porto Alegre: PPGC da UFRGS, 2024.

139 f.: il.

Thesis (Master) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2024. Advisor: Cláudio Fernando Resin Geyer; Coadvisor: Julio César Santos dos Anjos.

I. Resin Geyer, Cláudio Fernando. II. Santos dos Anjos, Julio César. III. Título.

“We build too many walls and not,
enough bridges.”
— Isaac Newton

AGRADECIMENTOS

Agradeço a Deus que iluminou o meu caminho durante toda essa jornada, me dando inspiração para persistir nesse programa de pós-graduação em computação.
Agradeço a minha mãe Helenice e meu pai Paulo Cesar que de forma especial e atenciosa me deram força e coragem em momentos difíceis.
Agradeço a cada integrante do Grupo de Processamento Paralelo e Distribuído (GPPD): Prof. Dr. Claudio R. Geyer pela orientação, paciência que teve comigo, contatos que disponibilizou com outros pesquisadores. Agradeço ao meu co-orientador, Dr. Dr. Julio César S. dos Anjos, pela brilhante orientação, paciência, que por muitas das vezes me motivou com sábias palavras, fazendo com que eu melhorasse não somente enquanto pesquisadora, mas também como pessoa. Aprendizados que levo para vida.

A lista sempre é extensa quando vencemos etapas importantes na vida, mas não poderia deixar de agradecer ao MsC. Breno Fanchiotti Zanchetta pelo imenso suporte refinando o texto final e por trazer energias quando não tinha. Ao Dr. Kassiano Matteussi pelos suportes técnicos ao longo da jornada. Shirlei Oliveira, Germano Bertoncello, Matheus Orlandi e Gerson Mayer pelas incontáveis conversas que tivemos à respeito das nossas pesquisas, desde o início da jornada. Aos amigos, além dos laços sanguíneos, que são presentes divinos na minha vida, Igor Balteiro e Iago Lima, me falta palavra para descrever a riqueza de sua essência.

Extendo meus agradecimentos à Universidade Federal do Rio Grande do Sul (UFRGS), pela excelência em pesquisas científicas, que é exemplo primordial da importância da educação no Rio Grande do Sul como também para toda comunidade científica além Brasil. Agradeço ao Intituto de Informática da UFRGS (INF) pela organização, determinação, e primosia com que tem auxiliado os estudantes de pós-graduação. Sem vocês a jornada da pós-graduação se tornaria muito maior.

RESUMO

A Internet das Coisas (IoT) vai além do uso de dispositivos inteligentes. Ela simboliza a transformação do mundo analógico para o digital, permitindo uma tomada de decisões mais eficiente. Para atender à crescente demanda por redes de sensores interconectados, processamento e análise de dados, é essencial implementar uma infraestrutura robusta. Esta infraestrutura deve ser capaz de fornecer respostas precisas aos usuários. O principal objetivo deste trabalho é investigar um dos desafios críticos da IoT: a latência de comunicação. O MiddleFog está situado entre os dispositivos IoT e a Nuvem, oferecendo uma camada dedicada para um melhor gerenciamento das mensagens de comunicação. Isso visa melhorar o desempenho e a confiabilidade da transmissão de dados entre diferentes protocolos. A arquitetura de middleware consiste em uma estrutura que permite a seleção dinâmica do protocolo de comunicação mais apropriado entre MQTT e CoAP, com base no estado da rede. Essa abordagem também minimiza as limitações de comunicação entre o MEC e a nuvem devido à latência, perda de pacotes e baixa eficiência da rede. Avaliações mostram uma taxa de perda de mensagens de 25% e uma melhoria de desempenho de 48% ao utilizar o MiddleFog na camada Fog.

Keywords: IoT, Middleware, Fog computing, IoT Protocols, Interoperability, Communication Protocols.

ABSTRACT

The Internet of Things (IoT) transcends the use of smart devices. It represents the conversion of the analogue world into a digital one, facilitating more effective decision-making. To support the growing demand for interconnected sensor networks, data processing and analysis, it is essential to implement a robust infrastructure. This infrastructure must be capable of providing accurate responses to users. In this context, cloud computing emerges as an efficient solution to satisfy the demands imposed by IoT. The main focus of this work is to study one of the most critical challenges of IoT: communication latency. MiddleFog is located between the IoT devices and the Cloud, i.e. the application at the Fog layer, providing a dedicated layer for better management of communication messages. In order to improve the performance and reliability of data transmission between different protocols. The middleware architecture consists of an architecture that allows dynamic selection of the most suitable communication protocol between MQTT and CoAP based on the network status. This approach also reduces communication limitations between the MEC and the cloud due to latency, packet loss, and low network throughput. Evaluation shows message loss rate of 25% and performance improvement of 48% when using MiddleFog in the Fog layer.

Keywords: IoT. Middleware. Fog computing. IoT Protocols. Interoperability. Communication Protocols. Sensor-based Technology.

LIST OF ABBREVIATIONS AND ACRONYMS

AMQP	Advanced Message Queuing Protocol
BLE	Bluetooth Low Energy
CDN	Content Distribution Networks
CON	Confirmable
DME	Device Manager Engine
DDS	Data Distribution Service
DSL	Domain Specific Language(
E2E	End-to-end Security
EC2	Amazon Elastic Compute Cloud
ECC	Ecliptic curve cryptography
GQM	Goal Question Metric
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IPSO	IP for Smart Objects
IoT	Internet of Things
MQTT	Message Queuing Telemetry Transport
M2M	Machine-to-machine
QoS	Quality of Service
REST	Representational State Transfer
RFID	Radio Frequency Identification
SLR	Systematic Literature Review
SQS	Simple Queue Query
SMQTT	Security Message Queuing Telemetry Transport
TAM	Technology Acceptance Model

WoT Web of Things

WMMS Web-of-things Mobility Management System

XMPP Extensible Messaging and Presence Protocol

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

Network connectivity is the backbone of the Internet of Things as it dynamically affects the way Internet of Things applications are designed for the Fourth Industrial Revolution [41, 44]. It seeks to provide an ecosystem that enhances process automation through connected intelligent objects, sensor networks, and devices that operate without any interaction with a human operator [60, 12, 70].

Internet of Things (IoT) technology provides a domain for interactions with the environment by offering an expansive ecosystem that is adept at capturing, processing, and transforming data. Through interconnected devices and sensors that are embedded in a wide range of objects, IoT facilitates the seamless exchange of information between physical and digital realms. These devices collect vast amounts of data from diverse sources, ranging from smart appliances in homes to industrial machinery in factories. Once captured, the data undergo sophisticated algorithmic processing that can leverage techniques such as machine learning and artificial intelligence to derive meaningful perceptions. Subsequently, IoT systems transform them into strategic action plans for, enabling enhanced efficiency, resource optimization, and improved decision-making in numerous sectors, including healthcare, transportation, agriculture, and other areas. Thus, IoT technology serves as a catalyst for innovation, can lead to smarter and more interconnected ecosystems.

Helpnet[1] provides a valuable illustration of the potential of the Internet of Things, and sheds light on its transforming capabilities across various domains. It believes that, the IoT has the capacity to revolutionize industry by enabling seamless connectivity and data exchange between devices, systems, and people. The article emphasizes the significance of IoT as a means of enhancing efficiency, productivity, and operational performance.

Tanenbaum [68] addresses several challenges in IoT, including massive connectivity, scalability, and securitys. This scenario requires a particular infrastructure so that it can play a critical role in a) meeting the high demand for reliability and efficiency b) processing data c) providing efficient communication in the real time d) establishing a system for offering feedback to the user.

IoT scenarios need open access and management models that take account of QoS parameters and low energy criteria [47]. This system has heterogeneous constrained de-

[1] <https://www.helpnetsecurity.com>

vices that adopt an integrated management approach with gateway, controllers, services, middlewares, and platforms to build an efficient and autonomous system. All these features are connected through communication protocols such as Zigbee, message queuing telemetry transport (MQTT), and Bluetooth Low Energy (BLE). The use of IP technology in embedded devices has recently been recommended by the IP for Smart Objects (IPSO) Alliance.

Some modern technologies, such as blockchain, machine learning, IoT, robotics, and 5G, require an extensive management of QoS parameters that is aimed at improving performance, reliability, and scalability optimization [47].

Bansal [10] provides a comprehensive analysis of contemporary technologies, together with their performance and scalability. He demonstrates how the middleware handles the task of storing all the data in the database server, which involves analyzing the data, making graphs and reports, ensuring security and privacy, and managing the entire system in which it is located. All of the data can be supported by Cloud, which is embedded within the middleware. Applications, such as smart cities and smart homes, rely on services and analytical information provided by this middleware. These services can be incorporated in an Application Programming Interface (API) format so that they can provide the entire IoT information and functionality to the IoT system, thereby enhancing the end-user experience.

Vaquero's [74] thorough examination of cloud computing technology provides a flexible and scalable environment that can be used for different channels, such as web and mobile applications. Protocols such as MQTT/SMQTT/AMQP, which were designed for PubSub, are highly significant and are currently being utilized for the integration of IoT applications across numerous domains, including financial services, process automation tools, transportation, and domain acquisition.

In the same area, building IoT software to support business needs in the described domains, requires protocols, while designing a suitable architecture plays an key role. In terms of protocol, HTTP is the most widely used application level protocol in the Internet. It is argued in recent studies that HTTP will represent the narrow waist of the future Internet. During the past decade, its adoption has become widespread mainly fuelled by the expansion of its HTTP-based infrastructure, such as Content Distribution Networks (CDNs), proxies, caches, and other middle boxes. While HTTP remains a popular choice for IoT communication, recent research suggests that alternative protocols, such as MQTT or CoAP, may offer better performance, scalability, and resource efficiency in certain use

cases [8].

1.1 Problem Statement

The ability to assess an efficient model for reducing the effects of latency and throughput on the communication protocols in the Internet of Things (IoT), depends on the network conditions between Edge devices and Cloud Computing. Researchers have introduced Fog Computing as an intermediate layer to mitigate the latency effects. Fog Computing extends the Cloud Computing paradigm to the Edge networking architecture by creating a wider range of applications and services [63].

The main feature of Fog Computing is low latency, efficiency, throughput, and reliable of data delivery service in a heterogeneous ecosystem [40, 47]. However, this device heterogeneity still leads to a reduction in data transfers because of the properties that the communication protocols include. This research topic is of value because of the convergence of factors such as performance and interoperability in Fog Computing.

The main problems in this area can be described as follows:

1. **Interoperability - Lack of standardization for IoT communication protocols:**
   The platforms come in various shapes and sizes and there are no standards for IoT [32, 48], although an attempt has been made to achieve uniformity [25]. When integrating all the IoT elements (systems, devices, services, middleware) in an ecosystem, protocols play an essential role in connecting the IoT devices and applications and achieving seamless interoperability. Most IoT platforms and architecture do not provide a single interface to abstract this complexity. The survey by Guthet *et al.* [33, 34] analyzes eight IoT platforms to simplify the descriptions of the IoT layer.

2. **Assessing the Performance Constraints of IoT Communication Protocols in Fog Computing:**
   According to the systematic review, the IoT communication protocols that are most often used are MQTT, CoAP, AMQP, DDS [56, 52, 50]. All the listed protocols sometimes experience problems with latency, and limitations of network throughput [19, 57, 39]. This represents a difficulty in maintaining a stable performance with constant gain, and affects the quality of service for end users.

3. **Quality of service (QoS) insufficient to the IoT metrics:**

Quality of Service (QoS) in the Internet of Things is a key factor because there are multiple IoT applications that have different patterns of behavior. In addition, there are different expectations with regard to quality of services in the same range, when they are considered more comprehensively. Manish *et al.* [61, 39] mentions three pillars: QoS parameter for Things, Communication, and the Network in the IoT ecosystem that is responsible for transporting data in close-real-time data and computing.

A persistent challenge in Fog Computing is how to enhance the efficiency, throughput, and data transmission reliability. Numerous studies have been conducted to assess the performance of various communication protocols, including MQTT, CoAP, HTTP REST (versions 1 and 1.1), DDS, and AMQP. The findings from these studies have pinpointed the optimal protocols for deployment across the Edge, Fog, and Cloud layers. Moreover, they provide a comprehensive performance assessment of the most commonly used protocols (MQTT, CoAP, HTTP REST - versions 1 and 1.1, AMQP), by offering valuable insights for optimizing data delivery in diverse computing environments.

However, HTTP REST protocol, particularly versions 1.0 and 1.1, has a serious drawback which lies in its inefficiency when transmitting small data packets intermittently. Establishing a TCP connection causes time delays and generates superfluous overhead. While HTTP relies on TCP for quality of service, and can ensure data delivery provided the connection remains stable, it lacks additional options for optimizing service quality.

Conversely, MQTT is able to achieve a commendable performance, especially in scenarios characterized by small message payloads and uncongested networks, and thus showcases its efficiency in specific conditions.

A significant drawback in the realm of IoT, as well as Fog and Edge computing, is the absence of standardized communication protocols. This lack of standardization means that the communication middleware does not possess the capacity to intelligently select the most suitable protocol from the prevailing network conditions, thereby affecting the overall efficiency and responsiveness of data transmission.

However, in the light of interoperability, performance, and quality of service concerns that can be found in Fog Computing environments, there is clearly an opportunity to make significant improvements.

1.2 Motivation

The development of MiddleFog is driven by a critical examination of the prevailing challenges in IoT communication, and is particularly concerned with interoperability, performance limitations, and the unacceptable quality of service (QoS) with regard to IoT metrics. The cornerstone of the factors that motivate MiddleFog's lies in the concept of interoperability that stems from a lack of standardization among the across IoT communication protocols. This fragmentation significantly hampers the seamless interaction between diverse IoT devices and systems, and leads to increased latency and reduced efficiency. By providing a middleware solution that can dynamically adapt to and integrate a range of communication protocols, MiddleFog seeks to foster a more cohesive IoT ecosystem, which can enhance device compatibility and streamline data exchange practices.

MiddleFog targets the performance limitations of existing protocols in the context of Fog Computing, and is thus pivotal layer that bridges IoT devices and cloud services. Fog computing is designed to process data closer to the source, and hence reduce latency. However, the performance of IoT communication within this paradigm is often undermined by the failings of current protocols, which are not optimized for the high-density, low-latency requirements of Fog Computing environments. MiddleFog adopts an innovative approach to re-engineer communication workflows and protocols, as a means of ensuring they are tailored to exploit the full potential of Fog Computing, and as a result, significantly enhancing the overall performance of the system.

A key motivating factor for Middlefog is its desire to improve Quality of Service (QoS) in IoT communications, as well as to meet and surpass IoT-specific metrics. Current QoS models are often inadequate for the unique demands of IoT applications, which require not only high bandwidth and low latency but also extreme reliability and scalability. MiddleFog is designed to incorporate advanced QoS strategies that are closely aligned with IoT metrics, and this ensure that services can meet the stringent requirements of diverse IoT applications, from smart cities to healthcare.

In recent studies, informs that MQTT and HTTP REST [8] are the most widely used protocols in solutions for IoT. The main reason for this is that they are mature standards for IoT and more stable than other protocols. They are the protocols preferred by many developers in their Fog-Edge-Cloud computing implementations. On the other hand, research that has been carried out on the performance of communication protocols

(OSI layers - application and transport), has proved that the MQTT and HTTP REST protocols (V1 and V1.1) [8] are not performative and there are other protocols such as CoAP, DDS and XMPP that have achieved more satisfactory results in Fog and Edge environments.

The driving-force behind this work is to carry out an evaluation test the hypothesis that the operational performance of software in the Fog layer, it can have a direct relationship with the state of the network together with the specific operations of each IoT protocol - CoAP, MQTT – so that their accuracy and execution time can be compared.

In this evaluation, we intend to make use of an environment without any simulation in the three layers (Edge-Fog-Cloud) and that represents Fog Computing. We also seek to C create a middleware that observes network metadata so that it can automatically change the IoT protocol and quality service (QoS) as a means of ensuring a satisfactory performance and, delivering data with a low packet loss.

1.3 Goals and Contributions

The overall aim of this research is to concentrate on developing an interoperable middleware to improve latency and manage throughput capacity by dynamically choosing the best IoT communication protocol for network conditions. To achieve interoperability, account will be taken of the most widely used protocols such as MQTT, CoAP, MQTT, DDS, and HTTP will be considered to meet the demand of the IoT ecosystem [63, 39, 48, 25, 62]. A number of goals must be clarified to achieve this objective:

1. The State-of-the-art in IoT Communication Protocols;

2. Improving performance by referring to the following metrics: latency, bandwidth, and throughput;

3. Implementing a modular algorithm that checks the network conditions and the payload size of the message sent between Edge, fog, and Cloud node in order to dynamically decide which is the best performance protocol from a defined list that is drawn up to improve performance;

4. Developing middleware in the Fog layer with modular that has the performance algorithm. middleware means creating modules that group protocols of the same nature to make middleware more customised;

5. MiddleFog is a use case.

The main objective of this work is to develop a modular communication middleware tailor-made for Fog computing environments. This middleware is designed to assess the operational conditions and dynamically select the most effective communication protocol for any given scenario, with the aim of optimizing latency. It will improve interoperability within the integrated Edge-Fog-Cloud Computing framework, by acting as a versatile interface compatible with a wide range of protocols, including MQTT, CoAP.

1.4 Thesis Structure Statement

This master's thesis is structured as follows. Chapter 2 establishes the theoretical framework for the study and gives an explanation of the experiments that were carried out against the background of IoT Middleware, Communication protocols, and IoT protocol on the basis of a publish-subscribe and request-reply interaction model. The Chapter 3.1.4 examines related work, and provides an overview of the performance of protocols in the area of Fog Computing, as well as discussing the research methodology that is employed. In addition, there is a detailed analysis of selected articles that represent the state- of- the -art of improvements in middleware performance in Fog Computing, as well as an outline a systematic review of the literature. Chapter 4 discusses the model that is used to overcome communication problems, and define the middleware architecture and design. In Chapter 5 there is a description of the evaluation model, together with the results and analysis of the results. Finally, Chapter 6 discusses some research contributions and general observations, and makes some recommendations for future work. Appendix A sets out the different groups of experiments that were conducted, each designed to make it easier to understand middleware performance.

2 THEORETICAL BACKGROUND

This chapter provides an overview of Internet of Things, and is followed by an examination of the background of middleware, communication protocols and protocols for IoT, which form an integral part of survey. From this perspective, we provide a brief introduction and descriptions of these features in the following sections.

2.1 Internet of Things: An Overview

The Internet of Things (IoT) is one of the driving forces constantly generates data from network sensors, household appliances, cars, and other physical devices without requiring human-to-human or human-to-computer interactions. IoT is an original paradigm that was introduced by the British technologist Kevin Ashton in 1999 and involved a short range of sensors embedded in network-enabled objects or devices with RFID (Radio Frequency Identification) to integrate communication between people and devices [73].

IoT represents a transformative phase in the digital revolution, and marks the point where the physical and digital worlds converge in an interconnected network of devices. At its core, IoT is a vast network of objects and devices that are embedded with sensors, software, and other technologies with the aim of connecting and exchanging data with other devices and systems through the Internet. It encompasses a wide range of entities, from simple household items like thermostats and light bulbs to more complex industrial tools. The essence of IoT lies in its ability to bring greater intelligence to everyday objects, by making them smarter and more responsive to human needs, and thus achieve a level of smart automation and operational efficiency that was previously deemed unattainable.

One of the foundational pillars of IoT is its capacity to collect, transmit, and process data in real or close-real-time, by providing insights and enabling technologies that were not possible before. This is achieved through sensors and devices that gather data about their environment or condition, such as temperature, location, or movement. These data are then transmitted over the Internet to other devices or to a central system where they can be analyzed and acted upon. The ability to monitor and control devices remotely adds a new dimension to how we interact with our environment, and has led to innovations in home automation system, smart cities, healthcare, agriculture, and various other fields.

The synergy between the IoT and Fog Computing, alongside the critical role of

IoT protocols, represents a significant evolution in how data is processed and managed across interconnected devices. Fog Computing, which acts as an intermediate layer between IoT devices and the Cloud, facilitates the processing of data closer to its source, and is thus able to reduce latency, conserve bandwidth, and enhance overall system performance. This decentralized approach not only accelerates decision-making processes but also supports more complex and real-time analytics. IoT protocols, such as MQTT, CoAP, and HTTP, are an essential part of this ecosystem, since they employ standardized methods for secure and efficient data transmission between devices, Fog nodes, and Cloud services. Together, these technologies create a more robust, responsive, and scalable IoT framework, that can lead to smarter solutions in urban planning, healthcare, industrial automation, and beyond, while addressing the critical challenges of privacy, security, and data sovereignty.

2.2 IoT Protocols

IoT devices naturally have a restricted number of resource capabilities in terms of energy, memory and storage while coexisting in an ecosystem with multiple communications channels and different protocols. However, the increasingly stringent performance requirements of IoT services, especially in terms of latency and bandwidth, are challenging their deployment. In addition to the diversity, there are differences in the types of resources, sizes, schema of data and state of the network (i.e high / low data traffic) [49].

This communication environment poses problems that need to be tackled when expanding IoT, such as:

- Low Power - Technologies like Zigbee, Low power WiFi, Low Power Wide-Area Network (LPWAN), Neat Field Communication (NFC) and Wireless Sensor Network (WSN) need to reduce their energy consumption in the light of the constrained resources.
- Memory Loss - an intensive routing protocol mechanism is required to increase the speed.
- Internet Protocol Stack - Protocols has been defined at various layers of communication in physical, network, application layer. Each layer has a varied protocol stack.
- Network Capability - The transmission system or network should be able to collect

all the data from sensors even when there is high network traffic congestion.

A heterogeneous IoT landscape must adopt novel and reactive strategies for dealing with these issues. IoT has its own protocol stack, which is different from other protocol stacks like the OSI model and TCP/IP protocol stack. The IoT model protocol Stack converts the different layers of models to the Application layer (Protocols are CoAP, MQTT, AMQP, XMPP, RESTFUL), Transport layer (Protocols are UDP and DTLS), Internet layer (Protocols are RPL and 6LoWPAN) and Physical/Link layer (its protocols are IEEE 802.11 series and IEEE 802.15 series) [5].

In general, candidate communication protocols differ from interaction models: request-reply and publish-subscribe. The Request-reply is the basic model for a communication protocol, and is often especially found in client/server architectures where it provides data exchange data mechanisms for clients, in which the server is responsible for the management of exchange data. In protocols like REST HTTP (V1.0, 1.1 and V.2) and CoAP, there is client/server interaction.

In contrast, publish-subscribe which is generally found in event-based architectures. These provide a decoupled communication between producer and consumer data, which leads to distributed and asynchronous interactions.

In this scenario, three basic elements can be found: publisher, subscriber and broker. There are also protocols that suppoty different type of QoS policies like MQTT, AMQP, DDS. There are also protocols that support both request-reply and publish-subscribe models, like CoAP, XMPP and HTTP V2 (support server push options), as discussed later. There are protocols support both request-reply and publish-subscribe models, like CoAP, XMPP and HTTP V2 (support server push options) as discussed later.

However, some protocols are not suitable for the IoT ecosystem like Web Socket, which is designed for real-time scenarios where data is pushed around from server to web client in a simultaneous bidirectional form of communication [5]. By following these definitions, we are able to provide an overview of the most popular protocols and their core functionality. These approaches are discussed in detail in the subsections, where each of the protocols is described.

2.2.1 MQTT - Message Queue Telemetry Transport

MQTT[1] is a messaging protocol, (lightweight publish/subscribe messaging protocol) designed for M2M telemetry in low bandwidth environments. It was designed by Andy Stanford-Clark (IBM) and Arlen Nipper in 1999 with the aim of connecting Oil Pipeline telemetry systems via satellite and standardized in 2013 at OASIS [52]. It can be implemented on an embedded IoT device with limited processing capacity and memory resources [16]. MQTT is standardized by ISO (International Standard Organization) standard (ISO/IEC PRF 20922) and can be used in conjunction with TCP/IP [20]. MQTT follows a publish/subscribe o that it can be implemented in a flexible and simple manner. MQTT (running above the TCP protocol) is suitable for constrained devices that have an unreliable or low-bandwidth. There are specifications for MQTT: MQTT v.3.1 and MQTT-SN (also known as MQTT-S) [50]. MQTT consists of three key components: subscriber, publisher, and broker. It is useful device that register as a subscriber for the specific content in order to be informed by the central point (broker) whenever publisher disseminates information of interest [48].

2.2.2 CoAP - Constrained Application Protocol

CoAP was Designed by the Constrained RESTful Environments (CoRE) working group of IETF[2] [56] with RFC (Request for Comments) of 7252 and updated by 7959 and 8613 for standardization [20]. It is a web transfer protocol that is concerned with optimizing resource constrained networks often found in IoT and M2M applications. Moreover, it follows the RESTful paradigm and allows CoAP clients to use HTTP-like methods when sending requests. In other words, clients can use GET, PUT, POST, or DELETE methods to manage the URI for identifying resources in the network [22]. Similar to HTTP, one of its most defining characteristics is its use of the tested and well-accepted REST architecture. With this feature, CoAP is able to support the request/response. As CoAP to be a lightweight protocol, the headers, methods, and status codes are all binary encoded, and thus reduce the protocol overhead unlike many other protocols. It also runs over a less complex UDP transport protocol instead of TCP, which reduce the overhead further. CoAP relies on its second structural layer for reliability; this is called the message layer

[1] <https://mqtt.org>
[2] <https://www.ietf.org>

and is designed for retransmitting lost packets. This layer defines four types of messages: CON (Confirmable), NON (non-confirmable), ACK (Acknowledgment), and RST (reset) [48, 22].

2.2.3 AMQP - Advanced Message Queuing Protocol

This is an open standard protocol that follows the publish-subscribe paradigm defined by OASIS [51]. It is designed to enable interoperability between a wide range of different applications and systems, regardless of their internal design. Originally, it was developed for business messaging with aim of offering a non-proprietary solution that could manage a large amount of message exchanges might be made in a system in a short period of time. This AMQP interoperability feature [22, 24] is significant because it allows different platforms that are implemented in different languages, to exchange messages.

2.2.4 DDS - Data Distribution Service

DDS, which is defined by the Object Management Group (OMG) is designed for M2M communication. It integrates the components of a system and, provides low-latency data connectivity, extreme reliability, and a scalable architecture that both business and the mission-critical Internet of Things (IoT) applications need [77]. This data-centric connectivity middleware protocol provides a virtual concept called Global Data Space, which is accessible to all interested applications. It includes a) a publish/subscribe architecture, b) the contracts established by QoS, c) automatic discovery and d) configuration. Unlike some other publish-subscribe protocols, DDS is decentralized and based on peer-to-peer communication, and as such does not depend on the broker component [22]. One of the salient features of the DDS protocol is its scalability, which comes from its support for dynamic discovery. The discovery process, which is the outcome of the DDS built-in discovery protocol, allows subscribers to find out which publishers are present, and to specify the information they are interested in together with the defined desired quality of service, and enables publishers to publish their data [18]. DDS provides developers with a highly configurable middleware that can allow them to control the end-to-end Quality of Service (QoS) of the applications through a wide range of attributes [39].

2.2.5 EXMPP - Extensible Messaging and Presence Protocol

XMPP is a open standard messaging protocol formalized by the IETF. It provides communication and file-transfers among nodes in a distributed network supporting real-time communication using Extensible Markup Language (XML) technology based on TCP.[17] It is a text-based protocol, based on Extensible Markup Language (XML) that allows both client-server and publish-subscribe and interaction. It is designed to allow users to send messages in real time, in addition to acknowledging the presence of the user. This protocol allows instant messaging applications to achieve all their basic requirements, including authentication, end-to-end encryption, and compatibility with other protocols. Hurak [22] et. al published an article about the disadvantages of XMPP namely i) that its authorization takes a long time while the clients are requesting access to the server and using ii) the fact that XML Stanzas in communication causes extra latency and delays, because of the structural unit of the XML stanza.

2.2.6 HTTP - Hypertext Transfer Protocol

HTTP is the most widely used application level protocol in the Internet[3].It is the fundamental client-server model protocol used for the Web, running over TCP. Communication between a client and a server occurs via request/response messaging. Although TCP ensures the delivery of large amounts of data, which is an advantage in connections that do not have strict latency requirements, it causes problems in resource constrained environments [50]. One of the main problems is that the constrained nodes usually send small amounts of data sporadically, which means that setting up a TCP connection takes time and produces unnecessary overhead. With regard to QoS, HTTP does not offer additional options, but instead relies on TCP, which guarantees successful delivery as long as the connection is not interrupted [22]. The most widely accepted version of this protocol is HTTP/1.1, but the HTTP/2.0 allows a more efficient use of network resources and a reduced latency by introducing compressed headers, by means of a very efficient and low memory compression format, as well as by allowing multiple concurrent exchanges for the same connection [22, 66, 21]

[3]<https://www.sandvine.com/hubfs/downloads/archive/2014-1h-global-internet-phenomena-report.pdf>

2.3 IoT Middleware

The integration of Internet of Things (IoT) middleware into Fog Computing represents a pivotal evolution in how data processing and network management are handled in distributed environments. This background exploration delves into the foundational aspects of IoT middleware within the context of Fog Computing, and highlights its role, challenges, progress, and implications for the future realm of technology.

IoT middleware serves as an intermediate layer that facilitates communication, data processing, and the management of services between IoT devices and their applications. In the context of Fog Computing, this middleware plays a crucial role in enabling decentralized Computing resources to process data that is closer to the data source. This proximity to the Edge of the network is crucial for applications requiring real-time processing and analytics, by reducing latency significantly more than Cloud Computing models. Furthermore, IoT middleware in Fog Computing environments helps in managing the complexity and heterogeneity of devices, and ensures seamless integration and interoperability across various platforms and protocols.

One of the most challenging tasks when integrating IoT middleware with Fog Computing is ensuring the seamless scalability and management of vast networks of distributed devices. The middleware must efficiently handle not only the huge volume of data but also the diverse and dynamic nature of IoT devices and their connectivity. This requires sophisticated management strategies capabilities within the middleware to dynamically allocate resources, balance loads, and ensure resilient communication even in the face of network failures or aggregate fluctuations.

Middleware fog, or fog computing middleware, is a pivotal component in the architecture of Internet of Things (IoT) systems, designed to mitigate latency issues and enhance computational efficiency. Situated between the cloud and edge devices, middleware fog serves as an intermediary layer that processes and analyzes data closer to its source, thereby reducing the time and bandwidth required for data transmission to centralized cloud servers. This architectural strategy is crucial for applications necessitating real-time data processing and low-latency responses, such as autonomous driving, industrial automation, and healthcare monitoring systems. By leveraging localized computational resources, middleware fog addresses the inherent latency and bandwidth limitations of traditional cloud-centric IoT models.

The primary function of middleware fog is to facilitate efficient data processing

and communication within IoT ecosystems. This layer supports the offloading of computational tasks from the cloud to local fog nodes, which are typically situated at the network's edge. This decentralization not only diminishes the data volume transmitted over long distances but also enhances data privacy and security by retaining sensitive information within a localized context. Additionally, middleware fog is capable of providing context-aware services, dynamically adapting to varying network conditions and the specific requirements of different IoT applications. This adaptability ensures that IoT systems can maintain optimal performance despite the dynamic and often unpredictable nature of their operational environments.

The study of middleware fog encompasses various aspects, including its design, implementation, and impact on IoT system performance. Researchers focus on developing algorithms and frameworks that enable efficient resource management, data processing, and communication within fog environments. Key research areas include optimizing load balancing across fog nodes, enhancing fault tolerance and reliability, and ensuring seamless interoperability among heterogeneous IoT devices and platforms. Middleware fog also presents opportunities for exploring new paradigms in data security and privacy, given its role in handling sensitive information at the network edge. As IoT continues to expand and evolve, the academic exploration of middleware fog remains critical for advancing the efficiency, scalability, and security of next-generation IoT systems.

The introduction of Middleware would enable researchers and developers to configure and create new applications and also introduce new IoT devices into the ecosystem and would avoid low-level reprogramming of the entire IoT ecosystem. IoT middleware [64] can be broadly divided into three categories:

- Service-Oriented Middleware: This kind of middleware enables end-users and developers to add (and modify) IoT devices to the IoT ecosystem as services. It provides services like access control, storage management, and event processing engine.

- Cloud-Oriented Middleware: This middleware enables data to be collected and interpreted of data with ease. However, it restricts the growth of the ecosystem in terms of types of IoT devices. The security model in Cloud-oriented architecture is defined by the Cloud which is being used.

- Actor-Oriented Middleware: This kind of middleware is open source and is designed on a plug and play model. IoT devices can be added to the IoT ecosystem as

a plugin and when an IoT device is not required, it can be easily removed without affecting the IoT ecosystem. The security model considered here, is configurable by users through a plug-and-play mechanism.

Bandyopadhyay et. al have published a classified list of IoT Middleware based on the following functional components [64, 72] of IoT middleware: Interoperability, Device management, Platform portability, Security and privacy and Contextual awareness.

2.4 Fog Computing

Fog Computing, which is an architectural paradigm, has emerged as a critical extension of Cloud Computing, designed to bring computational resources closer to the Edge of the network. This proximity to data sources ranging from IoT devices to local Edge servers enables Fog Computing to offer reduced latency, enhanced bandwidth efficiency, and improved privacy and security for distributed applications. By decentralizing data processing and storage, Fog Computing is able to address the limitations of traditional Cloud-centric models, particularly in scenarios requiring real-time or near-real-time analytics and decision-making.

The genesis of Fog Computing traces its roots back to in the proliferation of IoT devices and the exponential increase in data they generate. As these devices become more ubiquitous, there is clearly a greater need to process data locally, rather than relying on distant Cloud data centers, becomes increasingly evident. This is especially true for applications require immediate responses, such as autonomous vehicles, smart grids, and health monitoring systems. Fog Computing provides a scalable and flexible architecture that allows this data to be processed efficiently at or near its point of collection, and thus significantly reduce the reliance on bandwidth and the need to deal with latency issues.

Moreover, Fog Computing introduces enhanced security and privacy capabilities by enabling data to be processed within a localized environment and protecting sensitive information from exposure to external threats. This localized data processing model is designed to implement robust security protocols that can meet specific application requirements and regulatory standards. Additionally, Fog Computing supports a more resilient and reliable network infrastructure by allocating processing tasks to multiple nodes. This distribution not only enhances system performance but also ensures services continuity, even in the event of individual node failures or network disruptions. As the digital land-

scape evolves, the role of Fog Computing in supporting the burgeoning growth of IoT applications and services is poised to become increasingly pivotal, since it offers a more efficient, secure, and responsive framework for data processing and analysis.

In Chapter 3.1.4, several concepts were outlined that will be applied throughout this work. The concepts relating to the Internet of Things, IoT protocols, and middleware enable the reader acquire a comprehensive understanding of their various types, definitions, features, characteristics, applications, and objectives. Certain middleware models are employed to design systems that encompass several factors. aspects. These designs will be discussed in the following chapter on related work.

3 SYSTEMATIC REVIEW & RELATED WORK

In this chapter, We review middleware, architectures, communication IoT protocols, functionalities, and performance outcomes. This analysis helps identify current research gaps in the Fog computing domain. we explore the related work surrounding Fog computing middleware, which plays a crucial role in bridging the gap between cloud and edge devices.

3.1 Systematic Literature Review

The following sections outline the methodological guidelines, followed by the classification of the research and approach adopted. The systematic literature review was conducted to examine related works that address the objectives, research questions, and strategies arising from the concepts of Communication and Middleware Protocols implemented in a Fog Computing environment for processing message data from Edge and Cloud. It also takes account of the heterogeneity of the material. After the research questions had been defined, it was possible to define the attributes of the metrics employed to extract performance data for middleware at the Fog layer. As well as this, a description was given of the architectures, protocols, and technologies used to establish middleware communication at the Fog layer, and develop middlewares for the IoT being employed in Fog Computing.

3.1.1 Research Objectives

This research seeks to identify the primary studies that investigate the IoT communication middleware that has been used in Fog Computing.

Fog computing systems have the advantage of enabling local recognition, and giving support for user mobility, real-time interactions, low latency, high scalability, and interoperability in a way that cannot be achieved by cloud computing systems. In the contrast, it has a greater reduced processing capacity than Cloud and is subject to numerous variable and unpredictable negotiations within the heterogeneous ecosystem of Edge devices. One of the unresolved issues in Fog Computing concerns the question of how to increase the efficiency of data delivery, the transfer rate, and reliability.

The system plays a special role targeting the protocol middleware ecosystem, since it is a real protagonist. At the same time, there is no standardization of communication protocols for IoT and Fog and Edge communication middlewares, and there is no kind of intelligence that can automatically select the most suitable protocol for the network conditions.

In a specialized study that examined the unresolved issues regarding the performance of communication protocols at the application and transport OSI layers, a scenario was created in which the most widely used protocols in the IoT ecosystem, were compared, including MQTT, CoAP, DDS, XMPP, and HTTP REST protocol V2. These concerned the protocols that are not as efficient for Fog and Edge environments, and yield have more satisfactory outcomes.

Furthermore, this review aims at obtaining a comprehensive overview of the metrics that can improve middleware by:

- Determining which metrics are most widely used to improve middleware performance: latency, bandwidth and transfer rate;
- Finding out if there is an algorithm that checks the network conditions and size of the message sent between Edge and Fog Node devices so that it can dynamically decide which is the best performance protocol to enhance performance, since it is a known benefit for middleware at the Fog level.
- Identifying which are the best technologies, methodologies and algorithms that can be used to develop middleware for IoT at the Fog level for IoT;
- Proposing an innovative resilient communication middleware for the Fog environment. The purpose of this is to check dynamic conditions for selecting the best protocol in a particular scenario to improve performance on the basis of the metrics established in this research study.

It is also advisable to summarise the terms that are most often used in the selected articles to determine academic research trends.

3.1.2 Research Questions

To fill the gap about the metrics and limits of the system, a number of key question have driven this research about communication middleware for IoT in Fog Computing &

Cloud Computing. These include the following research questions(RQ):

RQ1 What attributes of the metrics are being used to extract performance data for middleware at the Fog layer?

RQ2 What architectures, protocols, technologies and approaches are being used to improve performance in middleware communication?

RQ3 What are the middlewares for IoT that are being used in Fog Computing?

RQ1 identifies:

- Metrics used in the investigated studies about Quality of Service;
- Methods used to enhance metric visualization;
- Recommended guidelines most of then laid down in the investigated studies;

RQ2 identifies:

- Characteristics of the communication middlewares that have been used in Fog;
- Architectures most often used to build middleware for IoT such as (SOA, PubSub, REST);
- Protocols most used to build middleware for IoT as (CoAP, MQTT, DDS);
- Limitations and obstacles to maintaining middleware arising from IoT restrictions (memory, disk space);
- Libraries, frameworks, software languages most often used to develop middleware in the Fog/Edge environment.

RQ3 identifies:

- Description of an empirical middleware with an improved performance in communication;

The summary of this study together with the formulated questions and their motivations, are listed in Table 3.1.

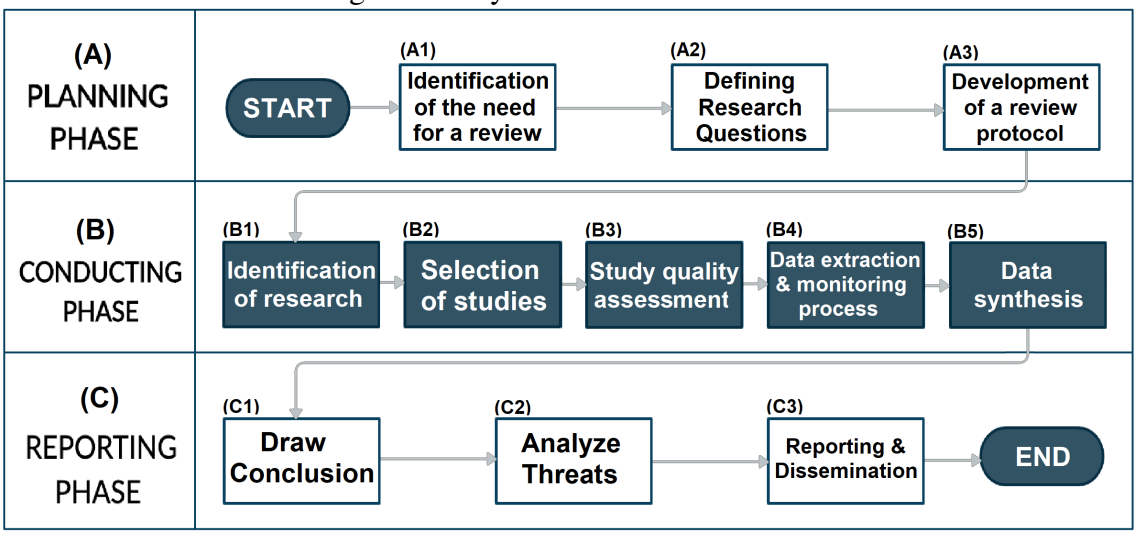
3.1.3 Research Methodology

This work follows the general guidelines and procedures described in [42, 7]. It involves searching academic publications in accordance with a protocol that is based on

Table 3.1: Research questions and motivation factors

RQ	Research Question	Motivation
RQ1	What attributes of the metric are being used to extract performance data for middleware at the Fog layer?	To determine the quality of Service (QoS) attributes, and guidelines applied in Fog.
RQ2	What architectures, protocols, technologies are being used to develop a middleware communication at Fog layer?	Identify main characteristics about architectures, protocols, technologies, difficulties/limitations for maintaining this middleware with regard to IoT restrictions (memory, disk space), elements to facilitate the development of middleware.
RQ3	What are the middlewares for IoT that are being used in Fog Computing?	To carry out an empirical study of the evidence that shows improvements in communication performance.

Figure 3.1: Systematic Review Phases



the systematic mapping process of Petersen et al. (2015), which is structured in three phases: (A) Planning; (B) Conducting; and (C) Reporting, as illustrated in Figure 3.1.

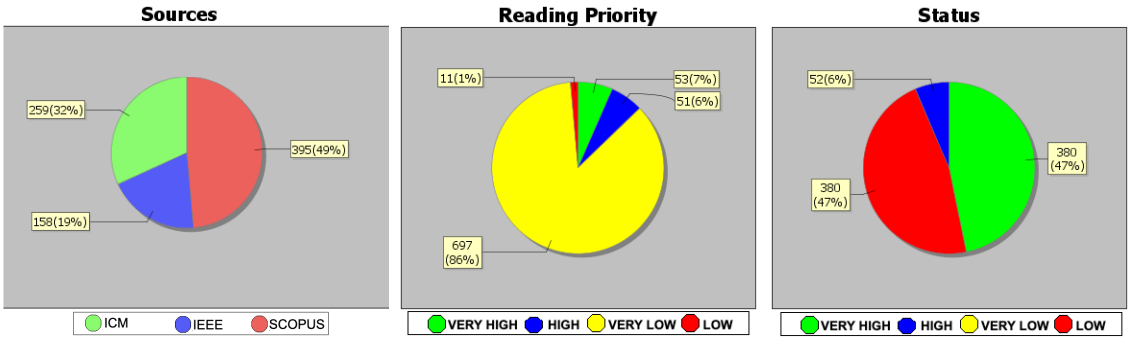
The Planning Phase (A) consists of three steps inside. The first step (A1) entails justifying the need for the review, and seeks to understand the features required for the review. The second step (A2) is to define the research questions, and formulates the need for a scientific inquiry. Finally, the third step (A3) involves developing a systematic review protocol, which is defined in the procedural rules laid down by SLR in order to process articles and surveys to obtain the state-of-the-art.

Since the formulation of the research questions drives the entire research methodology [42], they can be considered to be the most important part of any systematic review.

Thus, the Step A2 of the Planning Phase can be regarded as the most crucial part of this phase. After the final definition of the set of questions, the next step is to build search strings based on these questions to retrieve information from digital databases.

The results of the Planning Phase (A) retrieved 812 papers as illustrated in the sub-figure (a) of Figure 3.2, which included several formats (long, and poster papers) and duplicates in three different databases: *ACM Digital Library* [1], *IEEE Xplore*[2] and *SCOPUS*[3]. In addition, ACM and IEEE are recognised for their importance in the areas of computer science and electrical engineering; several other search engines have also been used, although the Science Direct [4] and Web of Science [5] are valuable resources for research, however the final result not presented new article, most of duplicate. For this reason, they was not include in this work.

Figure 3.2: Selection Stage:(a) Retrieve papers (b) Readings by priority (c) Accepted, Rejected and Duplicated papers



(a) (b) (c)

Furthermore, since each database has its own strategy for data mining, they require different ways of generating the search string for each database instance. The ACM database needs the string to be re-coded in the advanced search tool and restricts the scope of the search to records within the "ACM Guide to Computing Literature". In contrast, the IEEE database research was carried out by means of a standard metadata search. The strategy of SCOPUS was to consider articles and conference proceedings in the area of computer science. Presented in table 3.2.

There are tools designed to classify, rank and filter the most significant of the 812 papers in cost-effective time, and these support the systematic review process like

[1] <https://dl.acm.org>
[2] <https://ieeexplore.ieee.org>
[3] <https://www.scopus.com>
[4] <https://www.sciencedirect.com>
[5] <https://www.webofscience.com>

Table 3.2: Search String

Database	Search String Question
Scopus	TITLE-ABS-KEY (("iot" OR "internet of Things") AND ("middleware" OR "Protocols" OR "communication" OR "interoperability") AND "fog" AND "performance")
ACM	[[All: "iot"] OR [All: "internet of things"]] AND [[All: "middleware"] OR [All: "protocols"] OR [All: "communication"] OR [All: "interoperability"]] AND [All: "fog"] AND [All: "performance"]
IEEE	("iot" OR "internet of Things") AND ("middleware" OR "Protocols" OR "communication" OR "interoperability") AND "fog" AND "performance"

S.T.A.R.T. [6], and Parsifal [7]. The tool chosen, for this study was S.T.A.R.T. [53], because it offers valuable assistance in a number of areas, such as: i) learning disseminated topics; ii) defining theoretical approaches, applied methodologies, and bibliographical references; iii) detecting possible retrieved works that escaped the cope of the research findings; iv) highlighting innovative features in the user's work that challenge the state-of-the-art. Thus, this tool facilitates the process of scanning and filtering the papers in compliance with predefined criteria [37]. In opposition to this, Parsifal [45] was rejected since it does not contain several aspect of S.T.A.R.T. There are functionalities of [53], such as the Technology Acceptance Model (TAM), Goal Question Metric (GQM) and PICOC (Population, Intervention, Comparison, Outcome and Context).

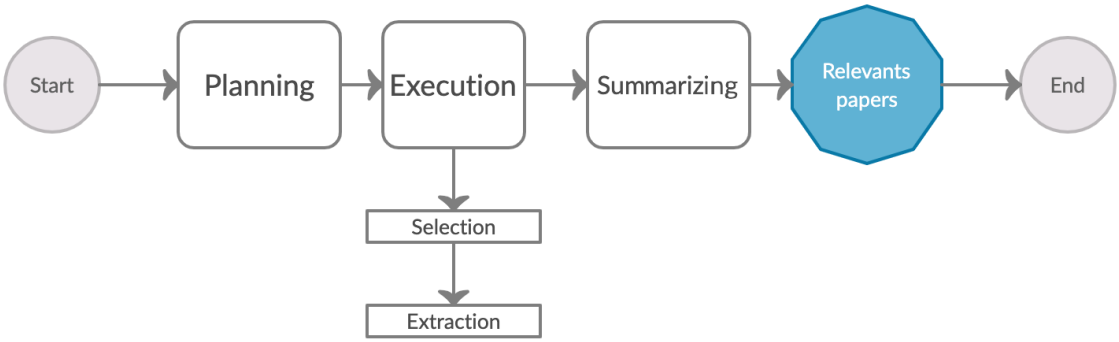
For this reason, all the sub-phases of the Conductivity Phase (B), i.e. B1, B2, B3, B4 and B5, and all the sub-phases from the Reporting Phase (C), i.e. C1, C2 and C3, are automated by the S.T.A.R.T. [53] tool (See phases in Figure 3.1). This automation begins with the intake of the 812 papers and ends with the 20 most useful papers, and is carried out by S.T.A.R.T. [53] in three stages, as illustrated in Figure 3.3: (1) Planning, (2) Execution and (3) Summarizing.

Essentially, the S.T.A.R.T.'s Planning stage serves merely to configure the main information that will be extracted from the 812 articles, such as: their goal, search problem, keywords, name of the targeted database name, type of study and inclusion/exclusion criteria. This Planning stage also includes some of the previously used configured specifications from Steps A1, A2 and A3 of Planning Phase (A) of Figure 3.1.

This is followed by the Execution stage of S.T.A.R.T. [53], which represents the scanning and filtering process that is carried out to find the most important papers. It

[6] <http://lapes.dc.ufscar.br/tools/start>
[7] <https://parsif.al>

Figure 3.3: S.T.A.R.T stages



contains two sequential and necessary steps: Selection and Extraction.

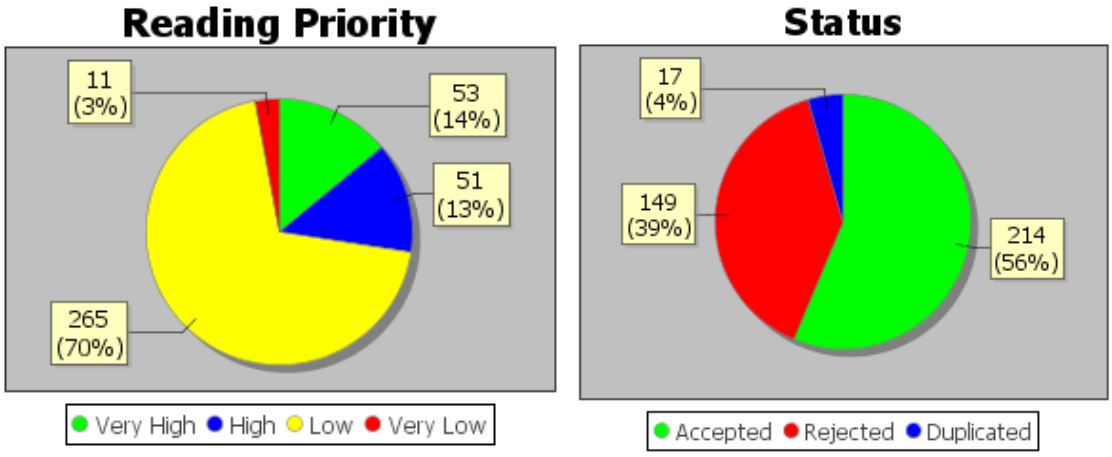
The Selection step starts importing all the files in BibTex/RIS and others formats, together with the metadata concepts. In this stage, 812 papers had already been retrieved from three different databases, as represented in Figure 3.2(a). The configuration defined in the planning stage, benefits automation by automatically ranking the papers through scanning and filtering operations, and giving them a score and suggests a priority to define which article to read (see Figure 3.2(b)). In this work, 53 papers were selected as most useful and 51 were given high priority, which means that the remaining 708 were considered to be unimportant for the final selection.

In parallel with the Reading Priority, the S.T.A.R.T. Selection uses the predefined configuration to divide the Status of these 812 works into three categories with regard to output: Accepted, Rejected or Duplicated (see Figure 3.2(c)). With the aid of S.T.A.R.T. 52 duplicated papers were detected which do not qualify as having an acceptable output. In addition, only 380 of the 812 papers can be considered to be important and acceptable.

The Extraction stage started by focusing on the 380 accepted papers from the previous Selection stage. A second rigorous scanning and filtering process followed in accordance with predefined criteria for the priority readings, namely relevance, title, abstracts, keywords or another metadata page. This process reduced these 380 papers down to 214 accepted, 149 rejected and 17 duplicated papers, as represented in Figure (b) 3.4. Coincidentally, just as in the Selection step, the Reading Priority output from the Extraction came to 53 and 51 articles with very high and high priorities, respectively. Thus, it is safe to say that Extraction aided the task of removing many articles that were at the end of the reading queue.

Before the final phase of S.T.A.R.T. [53], the tool was subjected to a manual filtering was required with the tool to reduce the 53 valid articles to a smaller fine-grained

Figure 3.4: Extraction Stage:(a) By priority (b) Accepted, Rejected and Duplicated papers



(a) (b)

set of relevant articles. This study carried out a quality assessment to measure the usefulness and importance of the studies that investigated these research questions. After this, it collected the required information to answer the research questions, as well as to analyze and summarize the results. The total of valid articles came to 20, as represented in the Figure 3.7.

The third operational phase of S.T.A.R.T. [53] is Summarizing, as seen in Figure 3.3. This last phase starts with the 20 accepted papers from the previous step and involves scanning each of the papers. This highlights key information, such as tht given by word-cloud 3.5, and the most commonly used words in the publication title or abstract. The Summarizing outputs concluded with several graphical metrics one of them being graphic visualization 3.6. This shows the frequency of the inclusion and exclusion criteria and helps in answering the research question that was defined in the sub-phase A2 of the Planning Phase (A) (see Figure 3.1).

Figure 3.5: A summarizing of Word Cloud from the most relevant articles

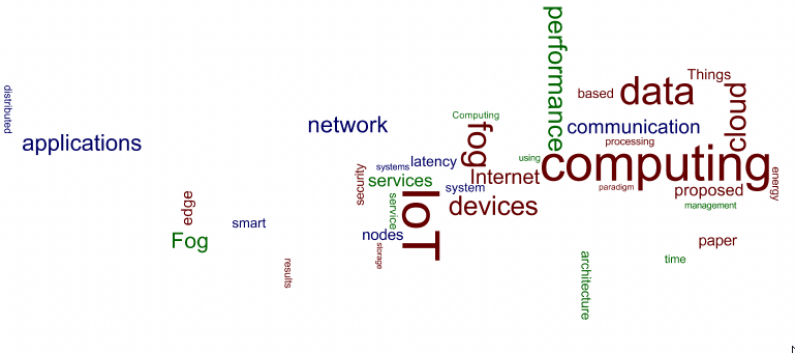
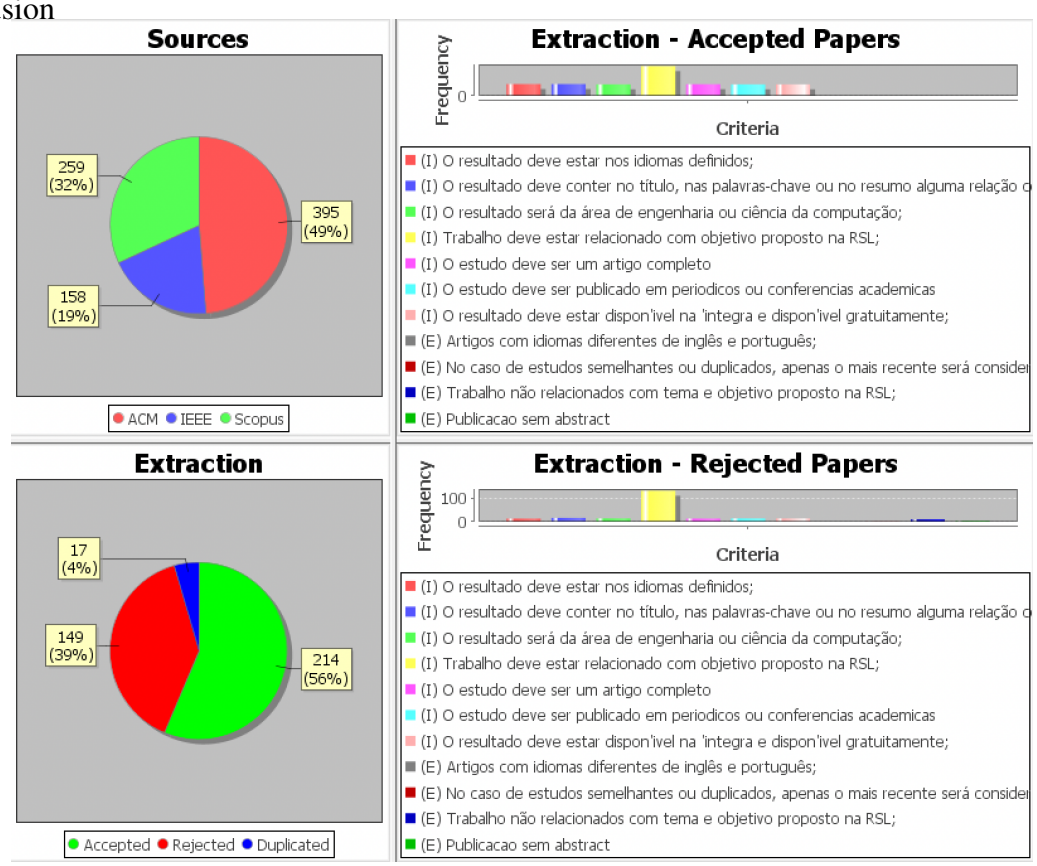


Figure 3.6: Graph Summarization with frequency of inclusion and exclusion criteria exclusion



Finally, since this thesis was first embarked on more works have been published added to the previous 20, thus making a total of 23 related works that can be seen in Section 3.2.3.

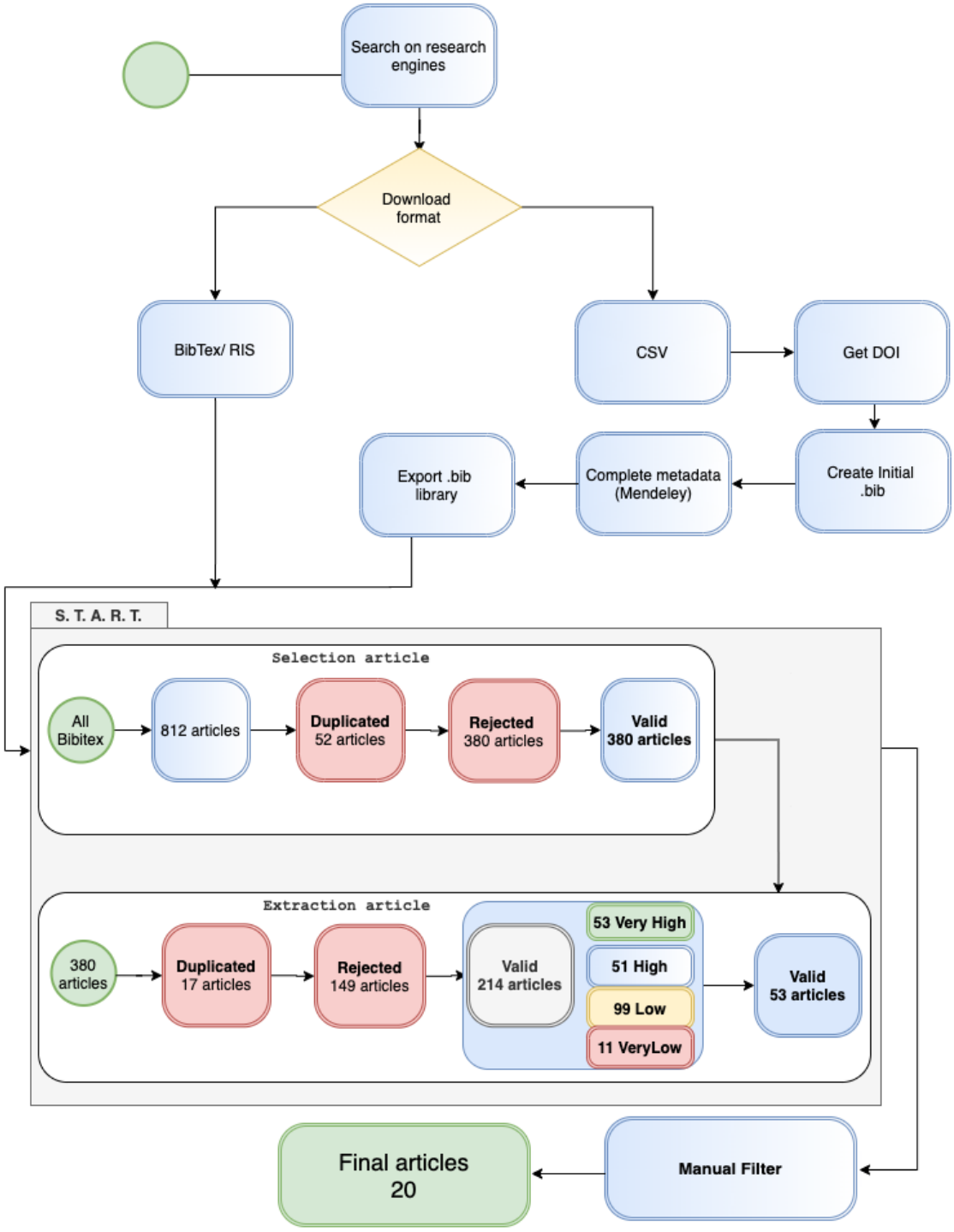
3.1.4 Research Results

This section classifies the different aspects of this work n accordance with the criteria that were applied to choose the study environment and its variables, but also to give an explanations of this experimental environment.

Addressing the research questions and presenting research interests relating to the communication middleware on Fog environment. The identification and presenting of communication protocols to the main terms explored in publications. We had address three questions (R1, R2 and R3), presented in the next chapter 3.1.4 and in this session we answer all of them.

Research is a formal and systematic procedure to what seeks to employ a scientific method [28]. Since the beginning of this research and throughout its progress, a number

Figure 3.7: Data Gathering and Processing Flow



of distinctive features have emerged as pivotal components of the conceptual framework. These entail the formulation of the problem, the formulation of the hypothesis, and the delineation of study variables, together with their correlation. It is thus essential to categorize these research features precisely in compliance with the rules that govern the scientific method.

The problem of the research and the theoretical overview must be addressed before attempting to classify its type. This type usually involves two factors: objectives and technical procedures.

With regard to objectives, this research can be regarded as exploratory and explanatory. It is exploratory insofar as it gathers bibliographical material to obtain a greater familiarity with the research problem, and thus be in a position to formulate it clearly. At the same time, this research is explanatory because it is able to determine which factors account for the occurrence of a particular phenomenon; moreover it gives the reasons and motivating factors that cause the events.

In addition, since this research study follows the the recommended guidelines of Gil [28] for classifying, research types on the basis of technical procedures, it can be regarded as bibliographic and experimental. It is bibliographic insofar as it is carried out on the basis of previously established material such as articles and books. It is also experimental because it defines the research object and variables that can influence patterns of behavior, as well as defining the mechanisms of control and observation that are required to study the effects of the variables on the object of study.

3.2 Related Work

We explore the related work, we delve into the research on Fog computing middleware, which serves as a vital link between cloud and edge technologies as result of systematic review.

3.2.1 Metrics Attributes: QoS Component

The rapid and emerging paradigm for connecting billions of physical objects, as well as empowering human interactions and improve quality of life (both physically and virtually), and thus introducing automation into the environment is called IoT. There is a sharp rise in the proliferation of IoT and it already covers many key domains and disciplines. In addition to the existing M2M, there are many other emerging technologies with empowering capabilities and functionalities.

The Quality of Service(QoS) metrics are being used to test for IoT communication middleware for Fog Computing. This metrics determines attributes, and guidelines

applied in Fog.

According to Gartner, *"worldwide was 15.1 billion in 2020 and is expected to rise to more than 29 billion IoT devices in 2030".* This reflects multiple IoT products and applications with different purposes that can provide a service quality in line with user expectations. Quality of Services is a means of managing system capabilities and its resources to provide IoT services. Its purpose is to obtain a clear idea of these services, as well as performance and the usability of the services to the consumers. The Quality of service metrics must first be defined and then the users will be able to understand how they can meet their requirements.

Manisha et al. [61] published a research study in which they describe the main QoS metrics that can help customers find the best IoT service for their application and then assess the optimization of service quality on the basis of the three components of IoT (Cloud, Fog and Edge), shown in Table 3.3.

Table 3.3: QoS for IoT Components for IoT [61]

Component	Description	Metrics
Computing	Compute and analyse the mined data collected from things.	Scalability, Dynamic Availability, Readability, Pricing, Response Time, Capacity, Security and Privacy, Customer Support Facility, User Feedback Reviews
Communication	Support the communication between the devices and with the external environment through the protocols used for communication in the network.	Jitter, Bandwith, Thoughtput and Efficiency, Network Connection Time, Financial Cost, Availability, Security and Privacy, Interoperability, Service Level Agreement, Monitoring, Readability
Things	The devices are embedded with the sensors and are capable of connecting to the Internet fals in the category of things. It can communicate to other entities at anywhere at any time.	Weight, Interoperability, Flexibility, Availability, Readability, Overall Accuracy, Long-Tearm Stability, Response Time, Range, Sensibility, Precision, Security, OTA Update, Power Consumption, Drift, Mobility Support

In the same way, Rameez et al. [40] define 19 attributes in 5 categories (Communication Attributes, Security, Connection, Operational, and Message and Payload support) These are the essential categories that are specifically designed for MQTT, HTTP, CoAP and XMPP protocols, as well as for IoT stakeholders, as shown in Table 3.4.

A final discussion among the authors resulted in 20 articles being selected as the

Table 3.4: QoS classification model [40]

Name	Description
Attributes	Communication Pattern, Transport Protocol, Multicast support, Reliability/QoS, Congestion control, Communication Complexity, Signaling Traffic Generated/Frequency of Updates, Connection establishment speed/performance, Session Orientation, Connection Security, Communication Security, User Security, Distributed Operation/Centralized, Service/Node discovery, Message retaining/durability, Caching, Message Overhead, Design Orientation, Fragmentation/Block Transfer
Categories	Communication Attributes, Security Attributes, Connection Attributes, Operational Attributes and Message and Payload support

most representative of the collection. The table 3.5 provides comprehensive information about regarding these articles.

The most common QoS attributes used in the selected publications, shown in Table 3.5 are: latency, response time, scalability, reliability, energy consumption, interoperability, security, and bandwidth. Any research findings with attributes or metrics that were not described, were not included in Table 3.5.

3.2.2 Communication IoT Protocols

Identify main protocols, technologies, difficulties/limitations for maintaining this middleware with regard to IoT restrictions (memory, disk space), elements to facilitate the development of middleware is one research question to map architectures, protocols, technologies are being used to develop a middleware communication at Fog layer.

In surveying the academic literature on communication and protocols for IoT, it was a noticeable feature that the IoT ecosystem provides heterogeneous smart and constrained devices with sensors or built-in wireless connectivity, actuators, and any other mechanism that can collect and transfer information to the network and then provide specific smart services[35, 41] that allow machine-to-machine (M2M) and human-to-machine (H2M) interactions.

The communication protocols used at the physical link layer for the IoT are WiFi, Bluetooth, IEEE 802.15.4, Z-wave, LTE-Advanced, Low-Power Wide-Area Network (LPWAN) and GPRS. Some specialized communication technologies are also employed like RFID, Near Field Communication (NFC) and ultra-wide bandwidth (UWB). RFID is the first technology that was used to apply the M2M concept (RFID tag and reader) [26]. When integrating all the IoT elements (systems, devices, services, middlewares)

Table 3.5: Selected Publications

Authors	Title	Year	Journals/Proceedings	Reference
Bansal, S., Kumar, D.	IoT Ecosystem: A Survey on Devices, Gateways, Operating Systems, Middleware and Communication	2020	Int J Wireless Inf Networks	Bansal et al. [10]
Kashif, H., Khan, M. N., Awais, Q.	Selection of Network Protocols for Internet of Things Applications: A Review	2020	Int Conf on Semantic Computing	Kashif et al. [41]
Dizdarevic, J., Francisco C., Admela J. and Masip-Bruin X.	A survey of communication protocols for internet of things and related challenges of Fog and Cloud computing integration	2019	ACM Computing Surveys	Dizdarevic et al. [22]
Bansal, S., and Dilip K.	IoT Application Layer Protocols: Performance Analysis and Significance in Smart City	2019	Int Conf on computing, communication and networking technologies (ICCCNT)	Bansal et al. [22]
Florea, I., Rughinis, R., Ruse, L., Dragomir, D.	Survey of Standardized Protocols for the Internet of Things	2018	Int Conf on Cont Systems and Computer Science (ICCCNT)	Florea et al. [9]
Ahmad, K, Mohammad, O., Atieh, M., Ramadan, H.	IoT: Architecture, Challenges and Solutions Using Fog Network and Application Classification	2018	Int Conf International Arab Conference on Information Technology (ACIT)	Ahmad et al. [1]
Corak, B. H., Okay, F. Y., Guzel, M., Murt, S., Ozdemir, S.	Comparative Analysis of IoT Communication Protocols	2018	Int Symposium on Networks, Computers and Communications (ISNCC)	Ahmad et al. [17]

in an ecosystem, protocols play a key role in combining the IoT devices and applications and supporting seamless interoperability. Internet Protocols, Network Protocols are rules for communication that allow the exchange of data (machine-to-machine and human-to-machine) when connected to the network [27]. The network protocols are divided in accordance with the nature of the service provided, and aligned with the layer where they are located on the Internet network, as shown in Table 3.6.

The application layer protocols for IoT have several ways of cooperating and achieving standardization [30]. Web-based applications and IoT applications are IP-based and rely on TCP and UDP for transport. These protocols handle information between gateways in the local network and Internet, have an update user with updated data and is-

sue commands from application to end devices [25, 9]. Various services can be provided to the users by communication protocols like MQTT, Constrained Application Protocol (CoAP) and Data Distribution Services (DDS), as shown in Table 3.6. Currently, there are different IoT protocols available, some are openly and others proprietary. There is not a standardized protocol [25, 27]. However, a list of IoT protocols associated with the current nature of the protocol network is being compiled. There is also an OSI Model with a list of available protocols which can be used in IoT platforms or service discovery with a selection of items to help reduce configuration efforts.

Al-Fuqaha et al. [26] published a classified list of general categories of IoT protocols, namely: application protocols, service discovery protocols, infrastructure protocols as shown in Table 3.6.

Table 3.6: Standardization strategies in support of IoT

Layer Name	Layer Description	Protocols
Application	Provide the services that users need. Handle information between gateways on the local network and the Internet. Carry commands from applications to end devices.	DDS, CoAP, AMQP, MQTT, MQTT-NSS, XMPP, HTTP REST
Transport	Provides the functional and procedural means of transferring variable length data sequences from a source to a destination host.	TCP, UDP, DTLS
Network	Responsible for facilitating data transfer between two different networks.	6LoWPAN, IPV4/IPv6
Link	Responsible for facilitates data transfer between two devices on the same network.	IEEE 802.15.4
Physical/ Device	Physical equipment involved in data transfer, such as cables. Where data gets converted into a bit stream.	IEEE 802.11 series, IEEE 802.15 series, Z-Wave, LTE-A, EPCglobal
Routing Protocol	Specifies how routers communicate with each other in a distributed network	RPL
Service Discovery	Automated detection of devices and services in a network.	mDNS, DNS-SD

3.2.3 Fog layer: Strategy Architecture

Carry out an empirical study of the evidence that shows improvements in communication performance characteristics about architectures and different strategy to leverage the performance.

As recommended by Dizdarević [22, 1], each protocol has its main features, standardization status, interaction model, Quality of Service (QoS) options, transport protocol, and security mechanisms. It turns out that the communication protocols have different

interaction models: request-reply and publish-subscribe.

The Request-Reply Model allows a client to request information from a server that receives the request message, and then processes it and returns a response message. This kind of information is usually managed and exchanged centrally. The two most well-known protocols linked to the request/reply model are REST HTTP and CoAP. Protocols: REST HTTP, CoAP, AMQP, XMPP, HTTP2.0.

The Publish-Subscribe Model provides distributed, asynchronous, and loosely coupled communication between data generators and destinations. The solution appears today in the form of numerous publish-subscribe Message-Oriented Middlewares (MoM). Protocols: MQTT, CoAP, AMQP, DDS, XMPP, HTTP/2.0

A survey of all the elements of the IoT ecosystem helps in understanding the concept of IoT, as well as its architecture, devices, operating system, middleware and communication interfaces. Architectures for the IoT ecosystem act in response to requirements like multilayer, middleware-based, service-oriented etc. The significance of low-, middle- and high-end devices in IoT devices are explained in this work. All the smart devices at ground level have been compared in terms of capabilities like architecture, computation, memory, and communication interfaces which have been discussed. The Operational System facilitates the development and sustainability of IoT in combination with the requirements of the hardware and there is a discussion of different IoT OS with regard to the resource constraints.

A comparative overview with regard to factors such as kernel, scheduler, memory management, performance, simulator, security, and electric power is displayed in the Table 3.2.3. IoT platforms and middle- ware act as a bridge between devices and applications to support heterogeneity, scalability, security and highly complex computational capability. IoT middleware has been analyzed from a consumer-centric Cloud-based perspective to one that is light-weight actor-based, and heavy-weight service-based. Basic communication have been described that can support IoT. There is a discussion of low-power communication networks and protocols for all the layers starting from the physical layer and going on to the application layer, to ensure the data can flow freely. Security and privacy issues grew significantly in direct proportion to advances made in the networking and communicating sectors.

This opens up several issues since the rise in the number of devices, technological integration, increased traffic, data storage and processing, privacy and security, etc., have become key areas of research. Cloud computing is technology based and designed to

operate and integrate with other recent technologies such as big data. The technology of Cloud computing refers to the processing power of the data at the “Edge” of a network. Additionally, it could be said that Cloud computing operates in a “Fog” environment. The interplay between the IoT, big data analytics, and Cloud and Fog computing makes it an IoT ecosystem for tackling problems like mobility, availability, storage, computational capability. As technology grows, more challenges arise and these become key areas of research. For example, the dynamic environment of the IoT opens up unseen opportunities for communication that will change the perception of computing and networking.

Table 3.7: State-of-Art Performance Result

Author	Approaches: Microservices	Approaches: 3/4/5 Layers	Approaches: Event-Based	Approaches: Content-Based	Approaches: Subject-Based	Approaches: SOA	Approaches: Cloud-Oriented	Approaches: Actor-Oriented	Protocols: CoAP	Protocols: MQTT	Protocols: HTTP 1.X	Protocols: HTTP 2	Protocols: AMQP	Protocols: XMPP	Protocols: SPDY	Protocols: SCTP	Protocols: QUIC	Protocols: SMQTT	Protocols: SCTP	Protocols: SNMP	Protocols: NETCONF	Protocols: DDS	Layer: Cloud	Layer: Fog/Cloudlets	Layer: Things/Edge	Quality of Service(QoS): Latency	Quality of Service(QoS): Thoughtput	Quality of Service(QoS): Response Time	Quality of Service(QoS): Scalability	Quality of Service(QoS): Reability	Quality of Service(QoS): Energy Consumption	Quality of Service(QoS): Interoperability	Quality of Service(QoS): Security	Quality of Service(QoS): Bandwidth	Strategy: Best practises	Strategy: Comparative protocols	Strategy: Transport Protocol	Strategy: Communication Pattern	Strategy: Payload Support	Strategy: Multicast Support	Strategy: Energy Consumption	Strategy: Memory/CPU Evaluation	Strategy: Latency Evaluation
Luntovskyy[47]		x																					x		x	x	x								x									x
Rameez[40]									x	x	x			x										x			x		x	x					x		x	x	x	x				x
Ahmad[1]		x																					x	x		x					x												x
Shekhar[59]		x																						x	x	x	x		x													x
Soma[64]		x																					x		x	x	x		x		x											x
Talluson[67]		x		x																				x	x		x			x	x											x
Bansal[10]						x	x	x	x	x													x	x	x									x	x							
Mudassar[2]																								x		x									x	x						x	
Wiss[78]									x							x							x	x			x								x							x	
Santana[58]		x	x																				x	x											x	x							
Hinze[38]				x																			x	x		x																
Lopez[46]		x																					x	x			x								x							
Candanedo[62]																							x				x		x	x												x
Guth[31]		x									x													x			x		x	x												
Skabicki[63]		x							x											x	x			x			x		x	x					x			x	x			
Makkad[5]		x							x	x						x			x				x	x			x		x	x					x			x				
Carlucci[13]		x									x				x								x	x			x		x	x					x							x
Stenberg[66]		x										x												x			x		x	x					x							x
Colitti[16]		x							x		x		x				x						x	x			x		x	x					x							x
Florea[25]		x							x	x			x			x							x	x			x		x	x					x							x
Manisha[61]																							x	x		x	x		x	x					x							
Dizdarevic[22]		x	x						x	x	x	x	x				x					x	x	x		x	x		x	x					x							x
Banavar[8]				x																			x	x		x	x		x	x					x							x

4 PROPOSED MODEL

This chapter discusses the proposed architectural model, its components, and the MiddleFog ecosystem. The investigation begins with a comprehensive overview of MiddleFog in Section 4.1, followed by Section 4.2, which includes a full explanation of the components of the architecture. Subsection 4.3 is devoted to MiddleFog's important entity called Decision Maker Engine (DME). Finally, Section 4.4 environmental divided into layers.

4.1 Planned Architectural Model

The research study addresses the question of latency in the Edge-Fog-Cloud environment. This problem is a crucial factor that must be handled in an heterogeneous ecosystem, especially as it makes use of a common protocol across platforms. For this reason, the investigation of **MiddleFog** requires a use case middleware at the Fog/Cloud layer. This middleware seeks to leverage system interoperability on account of its multiple interfaces, and improve performance through a lower latency that originated in the automated method for changing protocols.

Latency, in the context of network communication, refers to the time delay experienced as data travels from one point to another. This delay can occur at various stages, including the initial signal transmission, the processing of data by intermediate devices, and the final delivery to the destination. The primary components contributing to overall latency are propagation delay, which is the time it takes for a signal to traverse the physical medium; transmission delay, the time needed to push all bits of a packet onto the transmission medium; processing delay, the time routers or switches take to analyze and forward data; and queuing delay, the time data packets spend in queues waiting to be processed.

Latency is a crucial factor in determining the performance of networked applications, particularly those requiring real-time interaction. For example, in online gaming, high latency can result in noticeable lag, affecting gameplay experience. Similarly, in video conferencing, excessive latency can cause delays between audio and video, disrupting the flow of conversation. Financial trading systems also depend on low latency to execute transactions swiftly, as even milliseconds can impact the outcome of trades. Therefore, minimizing latency is essential for enhancing the user experience and perfor-

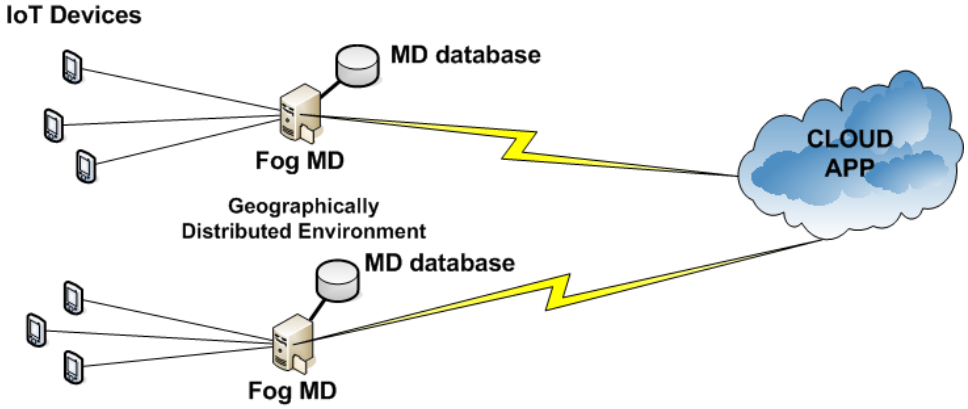
mance of various applications.

Various techniques and technologies are employed to manage and reduce latency. Optimizing network paths to avoid unnecessary routing, implementing content delivery networks (CDNs) to cache data closer to users, and utilizing edge computing to process data near its source are effective strategies. Edge computing, in particular, significantly reduces latency by decreasing the physical distance data must travel, thus enabling faster response times. As technology continues to advance, efforts to mitigate latency will be pivotal in supporting the growing demand for high-speed, real-time communication in diverse applications.

From an architectural perspective, the MiddleFog's software was designed for the Edge-Fog-Cloud computing ecosystem within a heterogeneous geographically-distributed environment. This diversified environment corresponds to the decentralized and heterogeneous collection of autonomous processors (nodes) communicating over a network with individual goals.

In the ecosystem mentioned earlier, the autonomous processors stand for the IoT devices (Edge) that carry out the local processing. At the same time, Fog nodes usually have more computing power than its Edge peers; this means, they are responsible for more middleware workloads and for storing temporary data. However, the Fog is not able to handle all high-demands in terms of computation and storage. In these, the tasks are scheduled to the Cloud. MiddleFog is located at the Fog level, so that it can assist in reducing the latency between Fog and the Cloud.

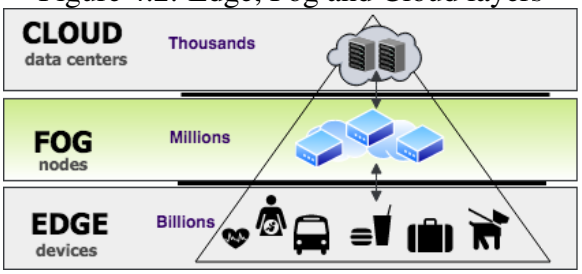
Figure 4.1: MiddleFog followings the three-layer architectural ecosystem



This environment of a three-tier architecture (Edge, Fog, and Cloud components - Figure 4.2). The concept of a three-tier model separates the system into three logical and physical computing tiers, which is the predominant software architecture for traditional applications [22, 8].

In addition, the three-layer architecture for Fog computing represents the Edge, Fog, and Cloud, where each layer is partitioned into domains presented in figure 4.2. MiddleFog stands out from this traditional approach thanks to its unique dynamic algorithm that seeks the best IoT communication protocol based on network conditions.

Figure 4.2: Edge, Fog and Cloud layers



The MiddleFog workflow follows defined stages in this ecosystem. It represents the three-tier - Edge, Fog, and Cloud components (Figure 4.2) in a distributed ecosystem (Figure 4.1), where the IoT-constrained device (Edge) is embedded with a specific protocol, and does not have enough resource management for computing data. Likewise, the IoT device is directly connected with the Fog node, the place located by the performative middleware client/server (MiddleFog).

The MiddleFog client/server module follows a sequence of steps at the Fog node component, namely:

1. Receiving data from the Edge, through a list of possible IoT communication protocols (CoAP, DDS, MQTT, AMQP);

2. Making decision about choosing the best protocol in the network conditions and message payload size. There is a component called Device Manager Engine (DME) with artificial intelligence, which, provides this information;

3. Parsing data fin compliance with the protocol structure/format rules;

4. Sending the data to the Cloud using the chosen performative protocol;

5. Receiving the data from the Cloud and passing it on to the Edge, in case it is needed.

The third step at the Fog node component is important, because it enables the communication protocol to be changed to improve performance. At the Cloud component, the MiddleFog server module follows sequence of steps:

1. Receiving the data from Fog node;

2. Computing and processing the data;

3. Creating metrics about performance as a part of the monitoring/observability in all the transactions;

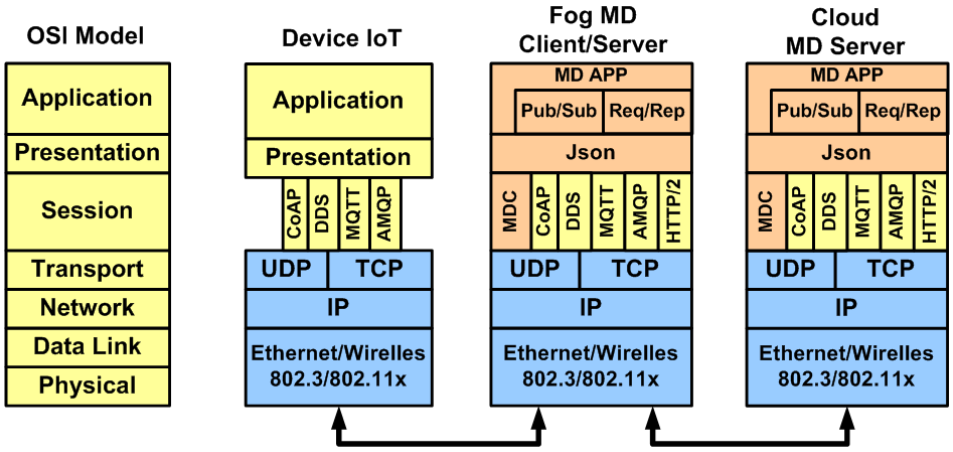
4. The Cloud server provides support for the most widely used IoT communication protocols to address interoperability;

5. Returning the requested data to the Fog node.

4.2 Architectural Components

MiddleFog was designed in a three-layered structure/ architecture: Device IoT (Edge), Fog Node with the Client/Server middleware module, and Cloud node with Server middleware module. Figure 4.3 shows the three layers represented following the Open Systems Interconnection (OSI) Model layer.

Figure 4.3: MiddleFog three-layer architecture represented in the OSI Model



- Data Link, Physical layers - this supports Ethernet/wireless with 802.3/802.11x specification

- Network layer - Supports the Internet Protocol

- Transport layer - Supports the Transmission Control Protocol and User Datagram Protocol

- Session layer

  This work does not implement any changes for level.

  - Device IoT - This supports the CoAP, DDS, and MQTT protocols;
  - Fog - This has the channel component to connect with the transaction AMQP & CoAP, DDS, MQTT, AMQP, HTTP/2;
  - Cloud - It has the channel component to connect with the transaction AMQP & CoAP, DDS, MQTT, AMQP, HTTP/2.

- Presentation
  - Device IoT - a constrained device is used to generate data from sensors in a protocol format;
  - Fog - Apply the Javascript Object Notation (JSON), the lightweight data-interchange format;
  - Edge - Apply the Javascript Object Notation (JSON), the lightweight data-interchange format.

- Application
  This work apply all changes at this level.
  - Device IoT - The Constrained device is used to generate data from sensors in a protocol format and process or send this data to another component;
  - Fog - Middleware Application with publish/Subscribe and Request/Reply proxies;
  - Edge - Middleware Application with publish/Subscribe and Request/Reply proxies.

The MiddleFog architecture which is designed in a three-layered architecture contains performative and interoperability modules:

1. **Edge layer - IoT Device**

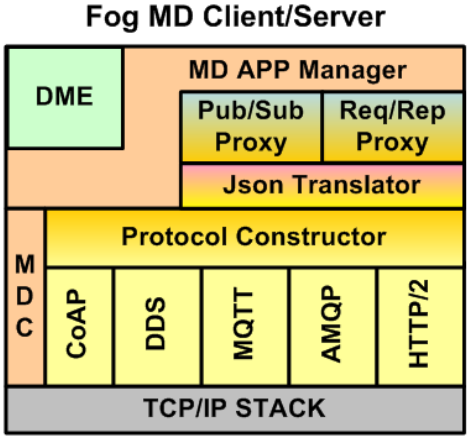
   This layer does not have any modules or make any contribution to this research, but is needed to intensify the valid IoT communication protocols, such as CoAP, DDS, MQTT, and AMPQ.

2. **Fog layer - Middleware Client/Server**

   The central part of MiddleFog is located in the Fog layer. It consists of the following components: Application Manager, Device Manager Engine (DME), Middleware

Channel (MDC), Proxies (PubSub and Res/Rep), Json Translator, Protocol Constructor, and IoT Protocols Communications, shown presented in Figure 4.4. Each component will be described in turn.

Figure 4.4: MiddleFog Middleware Client/Server architecture at the Fog layer



This module was designed to play two distinct roles: client and server.

- Client: Represents the bidirectional transactions between IoT devices (Edge) and Fog;
- Server: Represents the bi-directional transactions between Fog and Cloud.

(a) **Workflow - Middleware Client/Server:**

The Fog middleware will receive a request from the Edge device using one of the IoT communication protocols defined(MQTT, CoAP, AMQP, DDS). Following the sequence of steps:

i. This call will be intercepted by the *Middleware Channel (MDC)* responsible for making the connection to the corresponding proxy, based on the IoT protocol. It means protocols with Publish/Subscribe communication pattern will be redirected to pub/Sub proxy and protocols with Request/Reply communication patterns will be redirected to Req/Rep proxy;

ii. *Proxies Req/Rep and Pub/Sub* represent two different implemented and integrated protocols. MQTT broker implements MQTT and CoAP Server supports CoAP connections;

iii. The *JSON translator* supports the Req/Rep and Pub/Sub proxy structures. It understands both the metadata structures, extracts the main data and transforms in into a single and unique JSON format;

iv. *Decision Manager Engine (DME)* is a component designed to identify and apply best performative IoT Protocol communication, based on network conditions and message payload size;

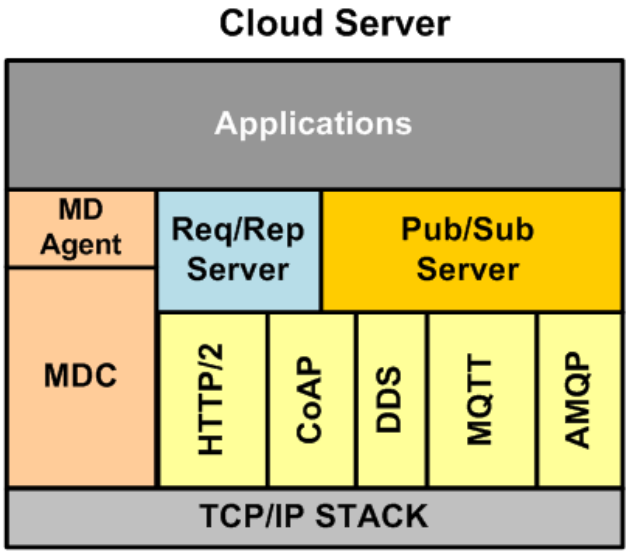
v. *Protocol Constructor* receives the data in JSON format from the JSON-translator component and converts it to an IoT communications protocol (MQTT, CoAP, AMQP, DDS, HTTP/2) structure defined by the Decision Manager Engine (DME);

vi. *IoT Protocol communication* (MQTT, CoAP, AMQP, DDS, HTTP/2) - This server will receive the JSON data in an appropriate format and protocol specifications. The is component has HTTP/2 to amplify its interoperability.

3. **Cloud layer - Middleware Server**

Module located at Cloud layer. This layer is consists of the following components: Agent, Middleware Channel (MDC), Servers (PubSub and Res/Rep), IoT and Communications Protocols, as shown in Figure 4.5.

Figure 4.5: MiddleFog Middleware Server architecture at the Cloud layer



The Cloud Server will receive the requests from Fog nodes using one of the IoT communication protocols defined (HTTP/2, CoAP, DDS, MQTT, AMQP). It follows a state sequence:

(a) This call will be intercepted by the *Middleware Channel (MDC)* responsible for making the connection to the corresponding server, through the IoT protocol. This means that protocols with a Publish/Subscribe communication

pattern, will be redirected to Pub/Sub server and protocols with Request/Reply communication pattern will be redirected to the Req/Rep server;

(b) *Proxies Req/Rep and Pub/Sub* represent different serve protocols, such as MQTT message broker that implements the MQTT and CoAP Server to support CoAP connections;

(c) *IoT Protocol communication* (MQTT, CoAP, AMQP, DDS, HTTP/2) this receives the JSON data in an appropriate format, together with the protocol specifications. The component has HTTP/2 to amplify the interoperability;

(d) *Middleware Agent (MD)*: this combines the constantly monitoring and observability agent to gather information about performance metrics, availability of systems, and logs.

4.3 Decision Maker Engine (DME)

This serves as the central element of the proposed model ecosystem. It is a module that operates in isolation, and is responsible for monitoring and analyzing messages in real-time within the network to determine the optimal IoT protocol (CoAP, MQTT, AMQP, HTTP2) in based on the network conditions.

The main advantages of the common middleware is that it enables the most suitable protocol to be selected and can give assurance of its capacity to build a module of software that can change the protocol automatically [69].

Several studies have been carried out that, comparing the performance of different protocols of communication such as MQTT, CoAP, HTTP REST - v1, V1.1, DDS and AMQP. Research articles have been published that can determine the optimal protocols for each layer, including Edge-Fog, Fog-Cloud, and Edge-Fog, as result of extensive performance testing and these has been widely disseminated (MQTT, CoAP, HTTP REST-V1/1.1, AMQP) [63, 13, 22, 25, 5].

The DME is required to specify the protocol requirements for extracting information from the network packet. The published studies related to Fog performance have and use of delay and total data (bytes) as a performance metric. Additionally, a performance test was conducted in 2018 using different sizes of data as payload (small, medium, and large) to measure latency, throughput, and error-rate. In the same way, it is recommended that studies should be published that are evaluated in terms of time taken and bandwidth

consumed for each payload transfer round-trip [69, 65].

From this perspective, the DME extracts data from the network package, by taking note of the following metadata:

- Message payload size;
- Analysis of the package to designed the lost one or, in some cases, not to measure the lost one;
- The fact that each message was sent or resent for each package from the network.

All of this information is important for defining the best IoT in a way that is based on the package metadata and network conditions.

In seeking to attain the objective of the DME, a series of tests were conducted in conjunction with the latest technology to understand the key parameters that were used. The baseline test plan validated the following scenario in a realistic environment:

1. A division of the message payload size into three distinct sizes, namely Small (20B), Medium (800B), and Large (2048B);
2. A configuration of the hardware Edge devices, namely NodeMCU/Raspberry Pi and Jmeter, by integrating the CoAP plugins and Meter;
3. The codebase is embedded in all the devices to send data with all payload message sizes in a random and automatic way,in a time interval ranging from 1 to 3 seconds;
4. An examination of the network protocol analyzer by means of using Wireshark;
5. Conduting a this test for CoAP with a small, medium, and large payload message size and QoS: Confirmable and No confirmation. See the diagram sequence for the CoAP Confirmable transaction, shown in the Figure 4.6;
6. Conducting the following test for MQTT with payload message sizes of small, medium, and large, and QoS of 0, 1 and 2. Please find the diagram sequence for the MQTT QoS 1 transaction, shown in the Figure 4.6;
7. After each run, extract the part of the information based on the metrics: package size, package loss, total time, average of total time, number of packages sent, rate (ms), burst rate, and burst start, shown in the Figure 4.7;
8. Compilation of the results for better decision-making, as shown in the Figure 4.7.

Figure 4.6: Sequential diagram - CoAP CON and MQTT QoS 1 transactions of baseline testing

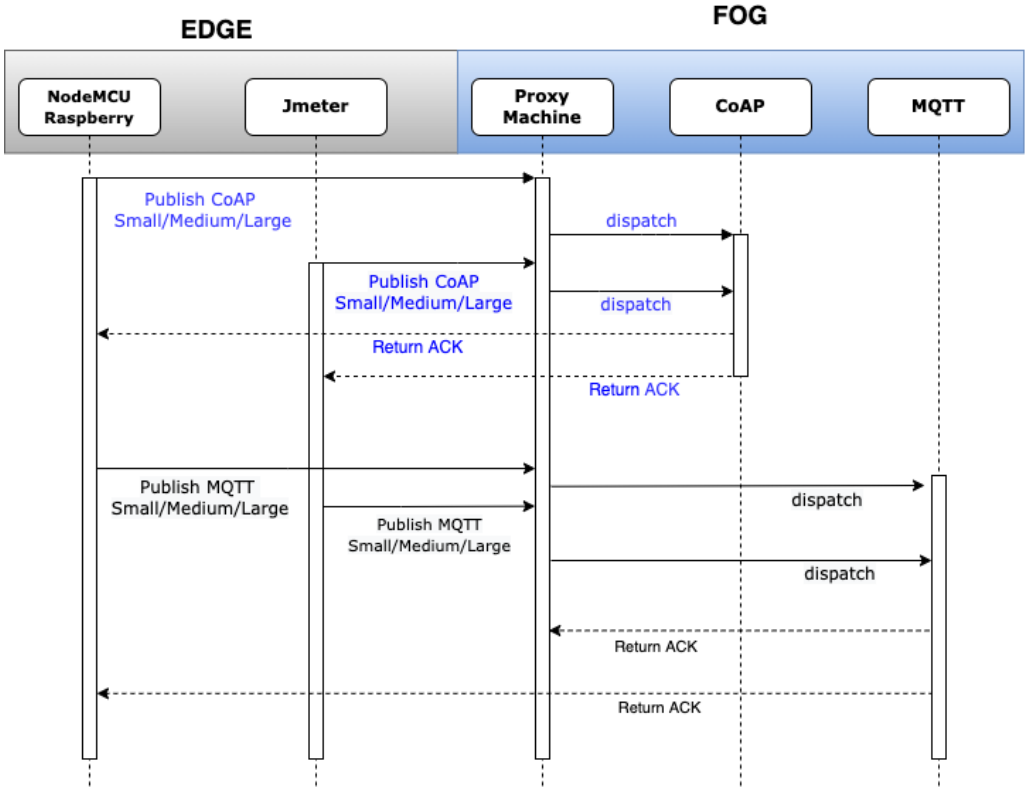


Figure 4.7: Compiled DME results table for MQTT/CoAP Message Size - Large(2048B)

	COAP		Message Size = 2048B					Package Loss	Total Time (seconds)	Media Total Time	%
			Payload size	Sent Packages	Rate(ms)	Burst Rate	Burst Start				
	CONFIRMABLE		340.12	320	0.0076	0.0800	1.070	0	41.90365	0.1309	100
	CON		616	160	0.0038	0.0400	1.071	0	35.369539	0.2211	84.41
	ACK		64.25	160	0.0038	0.0400	0.104	0	6.534106	0.0408	15.59
	MQTT		Message Size = 2048B					Package Loss	Total Time (seconds)	Media Total Time	%
			Payload Avr	Sent Packages	Rate(ms)	Burst Rate	Burst Start				
	QoS: 0		1060.73	83	0.0019	0.0200	17.928	0	33.70308	0.4061	100
	QoS: 0	Connected	103	1	-	0.0100	17.932	0	0.000151	0.0002	0.00
		Connected ACK	70	1	-	0.0100	13.462	0	0.012383	0.0124	0.04
		Publish	1097.5 (681*40+ 1514*40)	80	0.0019	0.0200	18.972	0	32.6751	0.4084	96.95
		Disconnected	68	1	-	0.0100	61.748	0	1.024007	1.0240	3.04
LARGE	QoS: 1		739.2	123	0.0028	0.0300	12.801	0	29.8741	0.2429	100
	QoS: 1	Connected	103	1	-	0.0100	11.783	0	0.000151	0.0002	0.00
		Connected ACK	70	1	-	0.0100	11.795	0	0.012383	0.0124	0.04
		Publish	1098.50	80	0.0019	0.0200	12.880	0	29.6204	0.3703	99.15
		Publish Ack	70	40	0.0010	0.0100	12.805	0	0.144096	0.0036	0.48
		Disconnected	68	1	-	0.0100	55.585	0	0.097105	0.0971	0.33
	QoS: 2		475.47	203	0.0046	0.0500	3.482	0	26.076454	0.1285	100
	QoS: 2	Connected	103	1	-	0.0100	2.359	0	0.000143	0.0001	0.00
		Connected ACK	70	1	-	0.0100	2.365	0	0.005642	0.0056	0.02
		Publish Message	1098.5 (683*40+ 1514*40)	80	0.0019	0.0200	3.482	0	24.795849	0.3099	95.09
		Publish Receive	70	40	0.0010	0.0100	3.486	0	0.141333	0.0035	0.54
		Publish Release	70	40	0.0010	0.0100	3.489	0	0.023386	0.0006	0.09
		Publish Complete	70	40	0.0010	0.0100	3.491	0	0.096856	0.0024	0.37
		Disconnected	68	1	0.0049	0.0100	46.284	0	1.013345	1.0133	3.89

After drawing up the baseline plan, conducting tests for all the protocols, and compiling the final results in the DME table, there are two complete tables:

1. The Table below shows the results from the CoAP and MQTT test, described in the baseline plan Steps 6 and 7 of the baseline plan. It is displayed as follows: 4.2 and 4.3.

Table 4.2 provides a visualization of the total time in seconds, package size, and number of generated packages. It combines the data from each protocol and quality of service based on the baseline tests. This information serves as the initial point of departure for the creation of the proportional Table 4.3, which yields the final value for use in the DME algorithm to determine the optimal protocol in the network conditions.

As a result of both tables, the input data for the DME Algorithm can be expressed as:

- Total time, package size and number of generated packages;
- The influence of QoS on the result: Latency, Jitter, Bandwidth, Throughput

2. The relevant metadata extracted from the network package, as shown in Table 4.1

   The key performance metrics are the message payload size and quality of service. As a result of the discovery process, the crucial performance metrics are significant metadata data from the network package, which aids the Decision-Making Engine in determining the optimal protocol based on the network conditions. For this reason, the payload message size and quality of service are used as input data in the DME algorithm and neither network traffic, latency, nor throughput metrics are used to extract data from the network as shown in Table 4.1.

Table 4.1: Key metadata extracted from the network packages.

NETWORK DATA	SCOPE	DESCRIPTION
Payload Message Size	**Yes**	Define the final number of packages that will be created to send over the network.
Quality of Service	**Yes**	Define the type and process between the client and server/broker.
Network Traffic	**No**	Not relevant to define the best protocol
Latency	**No**	
Thoughtput	**No**	

Algorithm 4.8 represents all the programmatic instructions for DME when receiving as input, the network to configure the best IoT protocol.

Table 4.2: Initial results with the aid of the key parameters

Total Time(seconds)			
Protocol - QoS	**Small - 20B**	**Medium(800B)**	**Large (2048B)**
MQTT - QoS 0	26.903072	15.346198	33.70308
MQTT - QoS 1	31.621311	26.435657	29.874139
MQTT - QoS 2	31.339829	25.306795	26.076454
CoAP - Confirmable	19.028300	28.112417	41.903645
Package Size(message + header)			
Protocol - QoS	**Small - 20B**	**Medium(800B)**	**Large (2048B)**
MQTT - QoS 0	98.81	825.33	1060.73
MQTT - QoS 1	89.47	461.51	739.2
MQTT - QoS 2	81.95	268.99	475.47
CoAP - Confirmable	91.50	481.50	340.12
Quantity of package Generated(unit)			
Protocol - QoS	**Small - 20B**	**Medium(800B)**	**Large (2048B)**
MQTT - QoS 0	43	43	83
MQTT - QoS 1	85	84	123
MQTT - QoS 2	166	167	203
CoAP - Confirmable	80	80	320

Table 4.3: Table with weighted average based on the Table 4.2

Total Time(seconds)			
Protocol - QoS	**Small - 20B**	**Medium(800B)**	**Large (2048B)**
MQTT - QoS 0	**0.85**	**0.55**	0.80
MQTT - QoS 1	**1.00**	**0.94**	0.71
MQTT - QoS 2	**0.99**	**0.90**	0.62
CoAP - Confirmable	**0.60**	**1**	1
Package Size(message + header)			
Protocol - QoS	**Small - 20B**	**Medium(800B)**	**Large (2048B)**
MQTT - QoS 0	98.81	825.33	1060.73
MQTT - QoS 1	89.47	461.51	739.2
MQTT - QoS 2	81.95	268.99	475.47
CoAP - Confirmable	91.50	481.50	340.12
Number of packages Generated			
Protocol - QoS	**Small - 20B**	**Medium(800B)**	**Large (2048B)**
MQTT - QoS 0	43	43	83
MQTT - QoS 1	85	84	123
MQTT - QoS 2	166	167	203
CoAP - Confirmable	80	80	320

Figure 4.8: Decision-Making Engine Algorithm

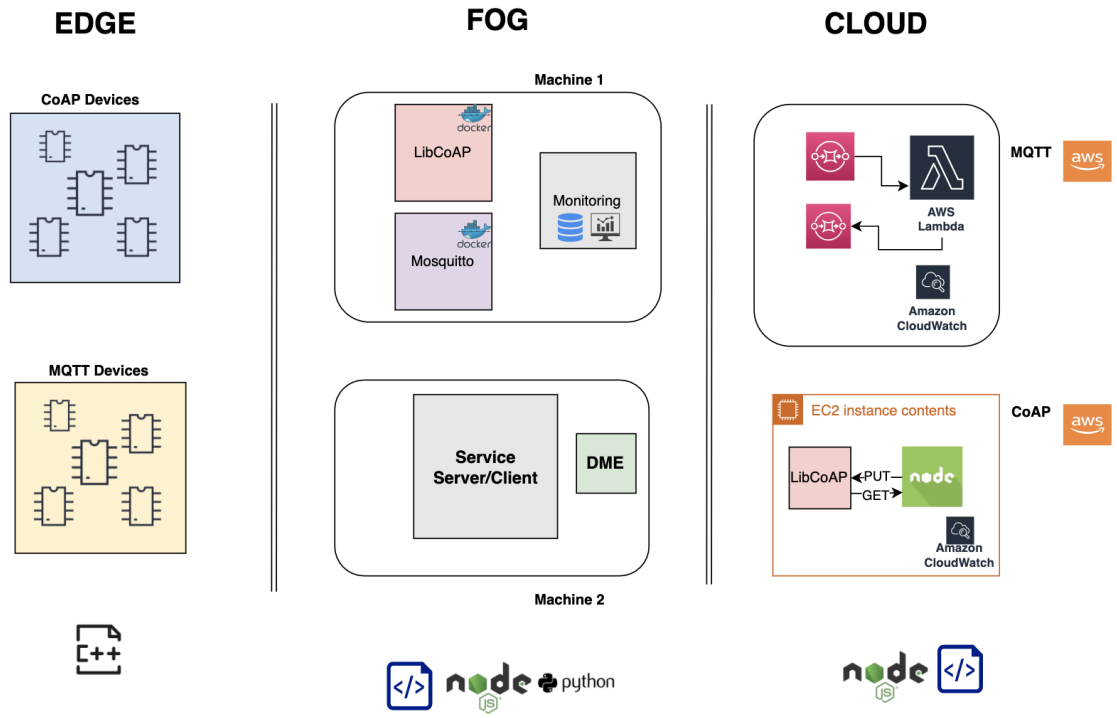
Algorithm 1: Decision Engine Maker Algorithm

```
Result: Decide the best protocol with performance gain
Execute Receive data extracted from the network;
small ← 100;
large ← 500;
finalSize ← "";
acceptablePackageLoss ← 2;
smallTotalTimeProporcionalLimity ← 0.60;
mediumTotalTimeProporcionalLimity ← 0.55;
bestProtocolArray ← [{small : coap}, {medium : mqtt0}, {large : mqtt2}];
while ThereIsNetworkData do
    lostPackage ← calculateLostPackage(networkData);
    packageSize ← identifyPackageSize(networkData);
    finalSize ← packageSize;
    if (packageSize ==' small' or 'medium') and
    (lostPackage ≥ acceptablePackageLoss) then
        totalTimeProporcional
          ← calculateTotalTimeProporcional(networkData);
        if (packageSize ==' small') and
        totalTimeProporcional > smallTotalTimeProporcionalLimity then
            finalSize ←' medium';
        end
        if (packageSize ==' medium') and
          (totalTimeProporcional > mediumTotalTimeProporcionalLimity
          then
            finalSize ←' large';
        end
    end
    changeProtocolTo( bestProtocolArray[ finalSize ] );
end
Method identifyPackageSize(packageSize):
    if packageSize ≤ small then
        size ← "small";
    end
    if packageSize > small and < large then
        size ← "medium";
    end
    if packageSize ≥ large then
        size ← "large";
    end
    return size;
```

4.4 MiddleFog Environment

This chapter describes the whole environment that is built to validate the proposed middleware model, **MiddleFog**. The middleware ecosystem contemples incorporates the three layers (Edge, Fog and Cloud), as illustrated in Figure 4.9.

Figure 4.9: MiddleFog Ecosystem - Three layers(Edge, Fog and Cloud)



- **Edge Layer** - This layer represents multiple devices from different manufacturers. To ensure a heterogeneous environment, the Nodemcu[1] physical devices codebase is written in the C/C++ language and configured with a unique IoT protocol (CoAP, MQTT, AMQP, HTTP2) and in parallel with a Jmeter emulated open-source application[2] which was used to increase the number of requests so that it could have an environment close to a real scenario and this emulator has been combined with a unique IoT protocol, like the physical devices.

- **Fog Layer** - This layer involves extending Cloud computing to the Edge of a network and the main contribution made by this work, **middleFog**. No Fog simulator was used, because the aim is to deal with real issues and calculate the solution for performance purpose. This layer includes:

[1] <https://www.nodemcu.com/index_cn.html>
[2] <https://jmeter.apache.org>

  - The docker machine with the IoT proxy protocols. This means all the brokers and servers can receive/send all the system data.
  - The docker machine with a monitoring service that is designed to store system data and log information.
  - The nodejs service with script technology, to apply the protocol changes to consumer and server data (server/client).
  - The Decision-Making Engine service, written in python language and libraries, responsible for finding the best protocol based on network conditions, and payload message size.

- **Cloud Layer** - This layer represents the on-demand high availability of distributed computer system resources, especially for storage and data processing. The whole cloud environment was built on the Amazon Cloud Infrastructure[3]. To achieve the support required for multiple protocols (CoAP, MQTT, AMQP, HTTP2) the Amazon Cloud infrastructure provides an interface to ensure address necessary interoperability. Each protocol has its own AWS Cloud environment, both isolated and independent.

4.4.1 Hardware Setup

The proposed **MiddleFog** incorporates three layers (Edge, Fog and Cloud) immersed in a heterogeneous IoT environment. It needs to have devices as the grounding for generating data for the system. Moreover, it contains a small set of hardware and a software emulator to increase the number of devices that generate data, as well as to address a real IoT scenario. The list of hardware devices is shown in Table 4.4 and the MiddleFog hardware in the Figure 4.10.

- **NodeMCU** is a low-cost, open-source Internet of things platform with ESP8266 Wifi, which comes from the Espressif Systems[4] manufacturer, and has hardware based on the ESP-12 module. Its features suggest it has easy adhesion to the IoT ecosystem. This work has 7 physical devices in which 4 boards were configured and embedded in the protocol MQTT and 3 boards were configured and embedded in the CoAP protocol. All the boards publish messages with a defined payload

[3]<https://aws.amazon.com/about-aws/global-infrastructure>
[4]<https://www.espressif.com>

Table 4.4: MiddleFog - Hardware Specifications for Edge-Fog-Cloud layers

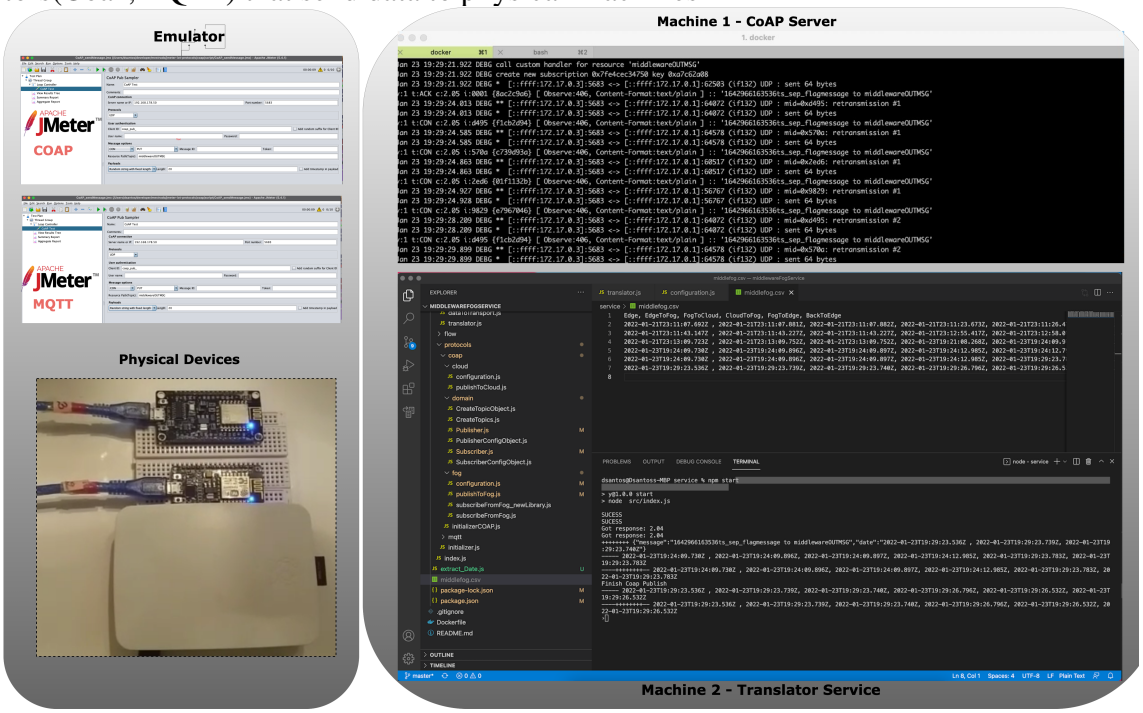
Hardware	Layer	Description
Physical NodeMCU, Raspiberry Pi Devices	Edge	Formed of 7 devices: NodeMCU ESP8266 CPU 32-bit RISC: Tensilica Xtensa LX106 - 80 MHz, Memory RAM 64 KB, Data 96 KB, External Flash QSPI 512KB/4 MB IEEE 802.11 b / g / n Wi-Fi, - 1 device Raspiberry Pi 3B+ SoC: Broadcom BCM2837B0 quad-core A53 (ARMv8) 64-bit @ 1.4GHz - GPU: Broadcom Videocore-IV, RAM: 1GB LPDDR2 SDRAM, Networking: Gigabit Ethernet (via USB channel), 2.4GHz and 5GHz 802.11b/g/n/ac Wi-Fi, Bluetooth: Bluetooth 4.2 (BLE), 3.5mm analogue audio-video jack, 4x USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI), Dimensions: 82mm x 56mm x 19.5mm
Physical Machines	Fog	Formed of 2 notebook: 8 GB 2400 MHz DDR4 / 2.2 GHz Intel Core i7 and 16 GB 2400 MHz DDR4 / 2.2 GHz Intel Core i7
Virtual AWS Machines	Cloud	AWS 2.8GHz Intel Xeon Cascade Lake Scalable CPUs

size(Small, Medium and Large), following the [8] experiment for their respective server or broker in a varied timeframe. The board is displayed in Figures 4.12 and 4.10.

- **Physical Machines** are 2 notebooks which seeks to host IoT protocol proxies(LibCoAP and Mosquitto), Decision-Making Engine service, monitoring service and a service to make the protocol changes. All these services, broker and server are operated the whole time to keep the main engines running and computing data in the ecosystem. The machine is shown in Figure 4.10.

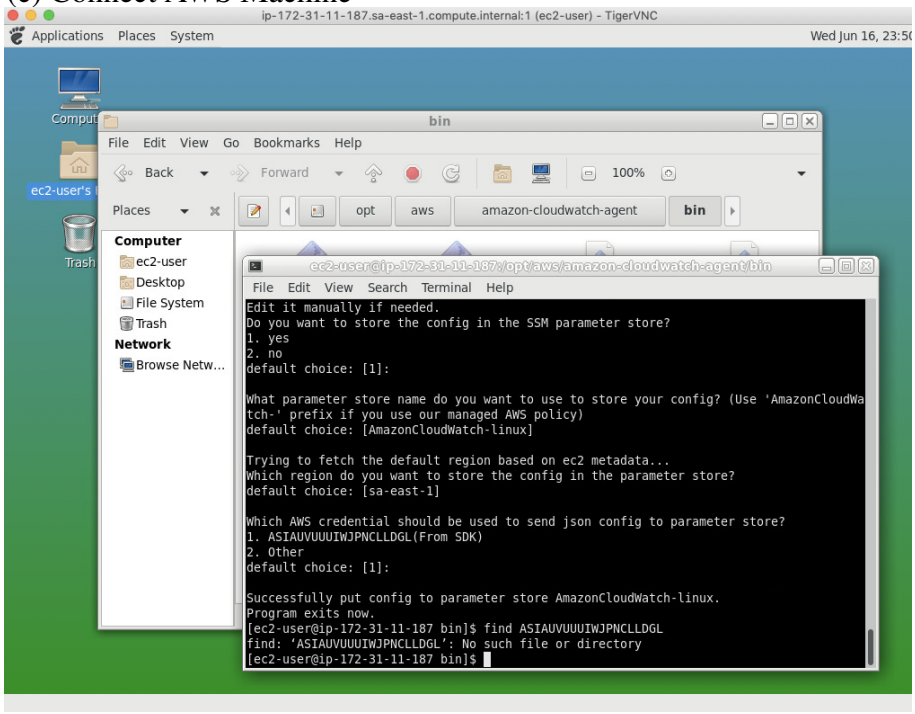
- **Virtual AWS Machines** are machines in the Amazon infrastructure which have elasticity of demand and host CoAP and MQTT services that are ready to use. The Cloud environment applied in the MiddleFog host has two services, one for each IoT protocol (CoAP and MQTT), to ingest and provide data from/to the Fog layer. The representation of both environments is shown in Figure 4.9 and the Figure 4.11 contains image (a) This reflects the process to setup a EC2 instance and includes the product and CloudWatch service so that it can note all the observability from the machine (logs, metrics, performance). Image (b) represents the CloudWatch

Figure 4.10: MiddleFog Hardware: NodeMCU, RaspberryPi and Emulators(CoaP,MQTT) that send data to physical Machines

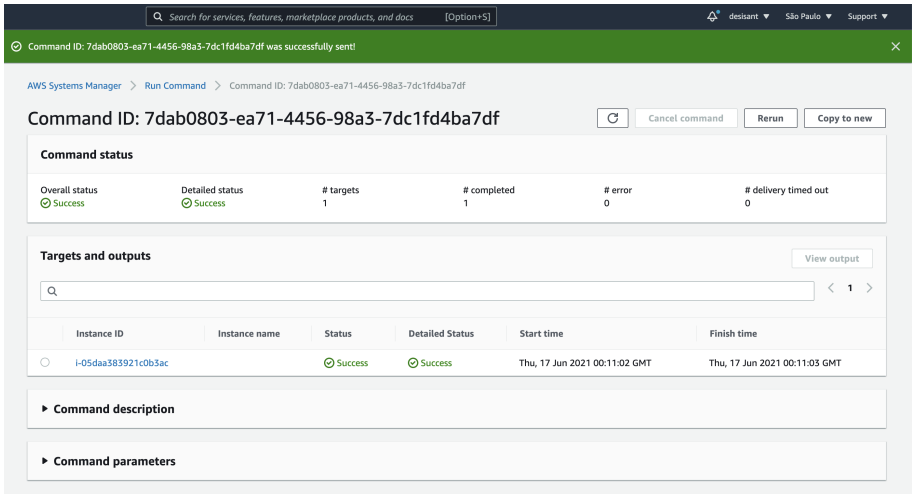


configuration applied to the EC2 machine after a few minutes/an hour.

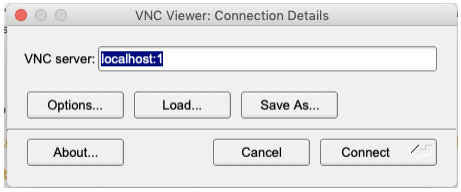
Figure 4.11: AWS Hardware Configuration: (a) Setup CloudWatch (b) Configured EC2 machine (c) Connect AWS Machine



(a)



(b)



(c)

4.4.2 Software Setup

The software stack **MiddleFog** embrace three layers (Edge, Fog and Cloud) which immersed in a heterogeneous IoT ecosystem. The Technologies such as language and, libraries were chosen on the basis of the state-the-art in Table 3.2.3. However, some tools or libraries like Pyshark, which is used in the proposed middleware, are not covered in the state-of-the-art. Thus, owing to the compatibility between parts of architecture, it is essential to have all the expected components for the architectural task. The list of software is shown in Table 4.12 and the MiddleFog software in the Figure 4.10.

The most noteworthy packages used in the board and, service configurations are listed with their versions and usage below in Table 4.5.

Table 4.5: MiddleFog - Software Specifications Edge-Fog-Cloud layers.

Software	Layer	Description
C/C++, Arduino IDE, Raspiberry Pi, Jmeter Emulator with CoAP/MQTT plugins	Edge	Embedded in all IoT physical devices - NodeMCU
NodeJS, Python, bash script, wiresharp, pySharp, InfluxDB, Docker, Mosquitto, Libcoap, Grafana	Fog	Technologies used in all the services and proxies machines at the Fog level
NodeJS, Bash script, Mosquitto, Libcoap	Cloud	Technologies used in all the services in Virtual AWS Machines

- **C/C++, Arduino IDE, Raspiberry Pi, Jmeter Emulator with CoAP/MQTT plugins** form the subset of technologies used at Edge layer. The software embedded in the physical device platform called NodeMCU boards were flashed by means of the popular Arduino IDE[5] and the codebase were written in C/C++ language. Having a successfully embedded process is necessary in a specific setup to allow Arduino IDE to flash the codebase on the NodeMCU board. In the same way, the Raspberry Pi board runs C/C++ codebase as a NodeMCU board. The number of physical devices is not enough to achieve an IoT scenario in which a large amount of data is generated per second, and on the basis of these criteria the **Jmeter emulator** address this need. Jmeter does not have the IoT protocol interfaces by default, and for this reason, two plugins were added, the plugin for CoAP is CoAP-Jmeter[6] and for MQTT is MQTT-Jmeter[7] as shown in Figure 4.12.

[5]<https://www.arduino.cc/en/software>
[6]<ttps://github.com/xmeter-net/coap-jmeter>
[7]<https://github.com/xmeter-net/mqtt-jmeter>

All the instructions and codebases on how to build the device were added to the Github repositories, shown below in Table 4.6.

Table 4.6: Github Edge Repositories

Github Repositories
Edge Device - <https://github.com/desireesantos/Edge-device/tree/main/protocols>
Jmeter Plugins CoAP/MQTT Protocols - <https://github.com/desireesantos/jmeter-iot-protocols>

- **NodeJS V12, python 3.8, bash script, wiresharp, pySharp, InfluxDB, Docker, Mosquitto, Libcoap, Grafana** form the subset of the technologies used in the Fog layer, as shown in Figure 4.9. In this layer, there are two machines, one is a docker machine with IoT proxy protocols monitoring service. This represents a machine with a message broker, MQTT Mosquitto[8], that implements the MQTT protocol versions 5.0, 3.1.1(the version used in the MiddleFog due libraries compatibility) and 3.1 and the libcoap[9] CoAP server version 4.2.0 that illustrates various server-side features to receive/send all the system data. The monitoring service is running to store system data and log information (Graphana[10] version 8.3.3 and InfluxDB[11]) version 1.8. The other machine is responsible for hosting two services. The first is the service written in node[12] version 16, JavaScript EMCS6 and bash script technologies, so that the protocol changes can be applied to consumer and server data(server/client). The second service is the Decision-Making Engine, written in Python language version 3.8 with Wireshark[13] version 3.6.1. and Pyshark[14],library version 0.4.3, which are responsible for identifying the best protocol in network conditions and payload message size. The technology applied in this layer is shown in Figure 4.12.

- **AWS EC2, AWS SQS, AWS CloudWatch, AWS Lambda, NodeJS, bash script, Mosquitto, Libcoap** form the subset of technologies used in the Cloud layer, shown

[8] <https://mosquitto.org>
[9] <https://libcoap.net>
[10] <https://grafana.com>
[11] <https://www.influxdata.com>
[12] <https://nodejs.org/en>
[13] <https://www.wireshark.org>
[14] <https://github.com/KimiNewt/pyshark>

Table 4.7: Github Fog Repositories

Github Repositories
Decision Maker Engine Service [branch DME] - <https://github.com/desireesantos/networkReader>
Service to apply changes - <https://github.com/desireesantos/middlewareFogService> Docker Container LibCoAP - <https://github.com/desireesantos/libcoap_container>

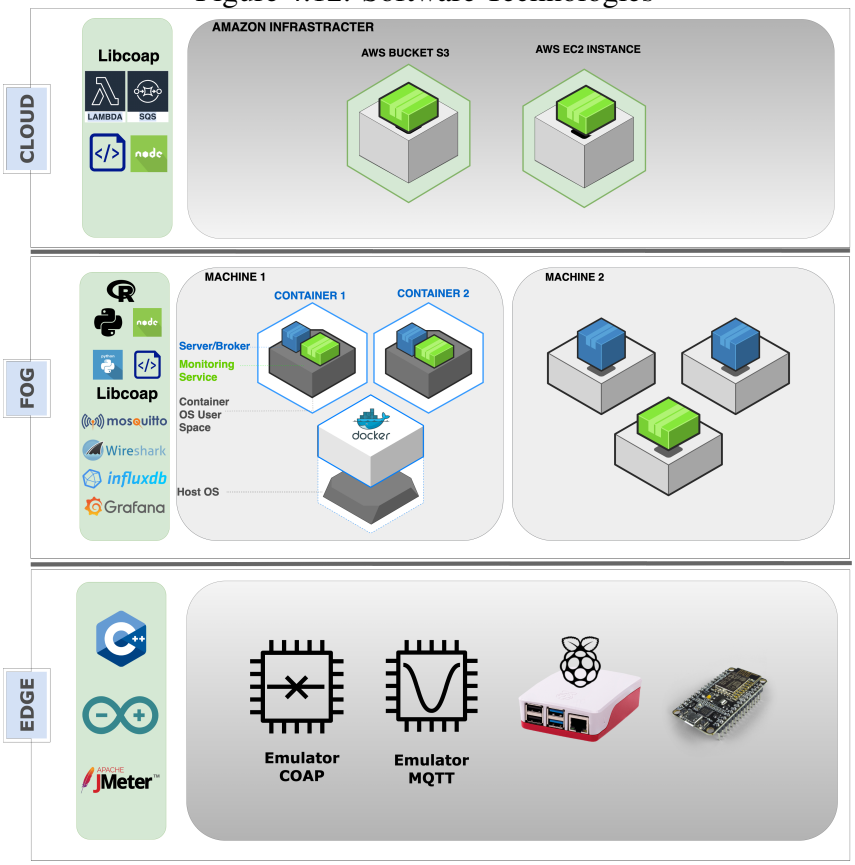
in Figure 4.9. In this layer, there are two environments within the Amazon infrastructure. The first is the CoAP environment on an Amazon EC2[15] instant content, where a service written in node version 16, JavaScript EMCS6 and bash script technologies are connected with libCoAP server version 4.2.0 to consume and provide data. In the same way, the MQTT environment has AWS Simple Query Service [16] as central messaging broker and is integrated with Amazon Lambda[17] to run codes without a provisioning or managing infrastructure and Amazon CloudWatch [18] for observability of its AWS service and applications. The technology applied in this layer is shown in Figure 4.12.

Table 4.8: Github Cloud Repositories

Github Repositories
Service Running at Cloud - <https://github.com/desireesantos/middlewareCloudCoAP>
Docker Container LibCoAP - <https://github.com/desireesantos/libcoap_container>

[15] <https://aws.amazon.com/ec2>
[16] <https://aws.amazon.com/sqs>
[17] <https://aws.amazon.com/lambda>
[18] <https://aws.amazon.com/Cloudwatch>

Figure 4.12: Software Technologies



4.4.3 Workloads

In this work, the MiddleFog experimental workload is formed, by describing details of the model and the configurations by testing the permutation of the IoT protocol. In the chart 4.13 Starting-point/ part of the system/all the flow.

The experimental workload, shown in the Figure 4.13 represents the IoT heterogeneous ecosystem where Edge device publishes data with different sized payload messages, quality of service, and variations of latency and time, as shown on the Table 4.9.

Table 4.9: Configurations that can be applied in the experimental workload

Category	Protocols	Values
Quality	CoAP	Confirmable
	MQTT	At most once (0), At least once (1) and Exactly once (2)
Payload Size	CoAP, MQTT	Small - 20B, Medium - 800K, Large - 2048B
Time	CoAP, MQTT	variation of 1s , 5s, 10s
Latency	CoAP, MQTT	variation of 5% - 30%

All these configurations are applied in the experimental workload, although each IoT protocol shares the same environment, but the data workflow requires different methods. This particular feature is shown in the Figures 4.14 and 5.1

Figure 4.13: MiddleFog Experimental Workload

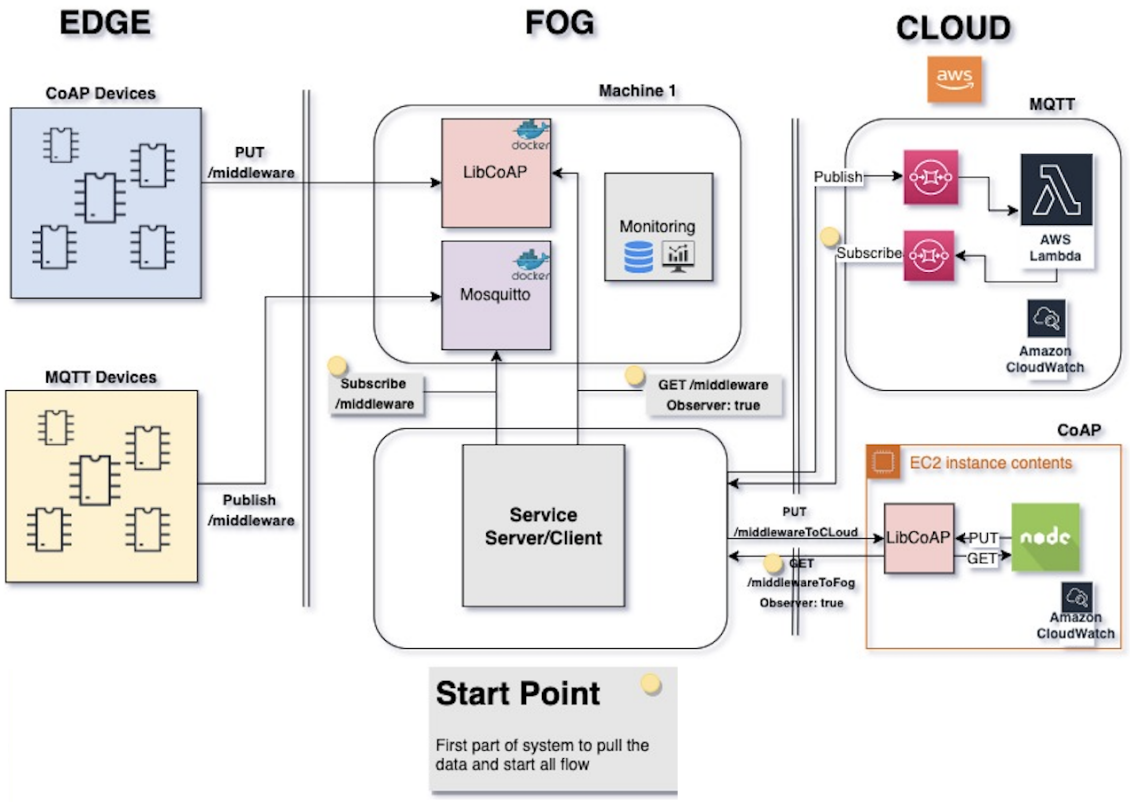
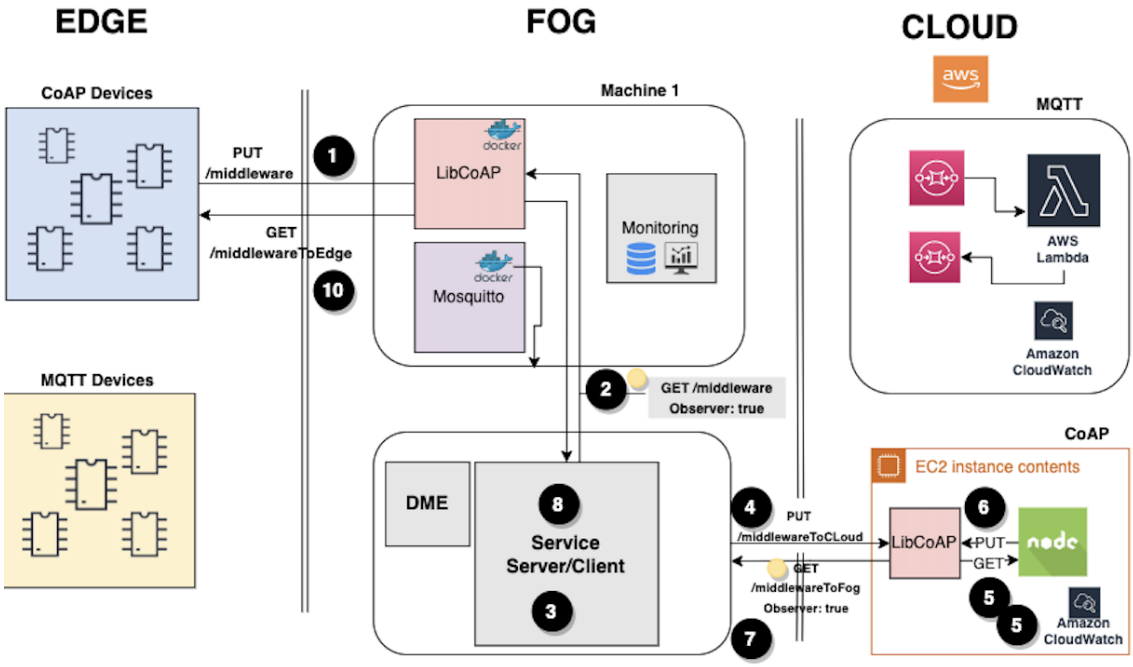


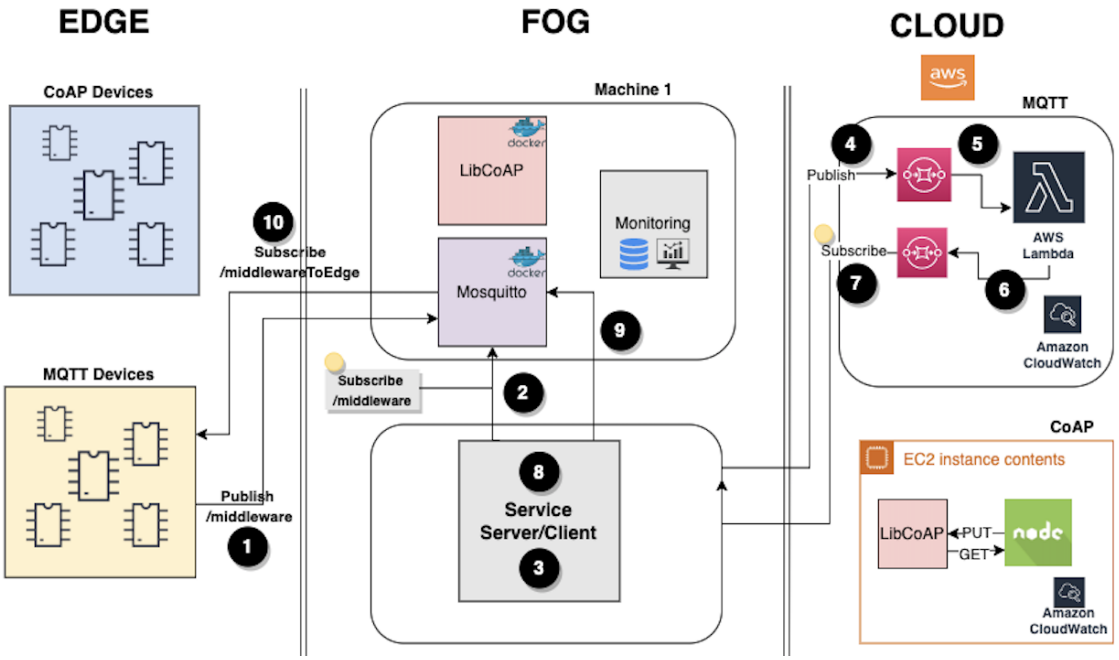
Figure 4.14: CoAP Data Workflow



The data workflow stages for the CoAP IoT protocol:

1. The CoAP devices sends a message with variations in payload size and time to the proxy at Fog level and the LibCoAP server receives the message created by Edge.
2. The service which is responsible for making protocol changes, observes the resource at Fog level to obtain all the new data created by Edge devices.
3. DME stages what is the best protocol, based on network conditions and payload size. The service gets and applies this information from DME. As well as this Besides that, the service processes the new data and packages in the format of the protocol.
4. The message is sent from Fog proxy to Cloud proxy using the IoT protocol defined by DME.
5. The proxy at Cloud receives this data from Fog.
6. The Service consumes the data posted by Cloud and delivers it to the CoAP server at Fog level.
7. The service assesses the resources to get all the data delivered by Cloud.
8. The service processes this data and sends it to the resource at Fog level.
9. Edge consumes this data.

Figure 4.15: MQTT Data Workflow



The phases of data workflow for the MQTT IoT protocol:

1. The MQTT device publishes a message with variations in payload size, quality of service and time for the proxy at Fog level and the Mosquitto broker receives the message created by Edge.

2. The service which is responsible for making protocol changes, observes the channel at Fog level to obtain all the new data created by the Edge devices.

3. DME states what is the best protocol, based on network conditions and payload size. The service notes the protocol definition and applies this information from DME. As well as this, the service processes the new data and package in the format of the protocol.

4. The message is published from Fog proxy to Cloud proxy using the IoT protocol defined by DME. The AWS SQS channel receives this data.

5. Later, after the message has arrived in the AWS SQS input, the AWS lambda function executes the defined instructions.

6. The AWS lambda function redirects the data to another output AWS SQS channel, and in this way the information is ready for the Fog to consume.

7. The service listens to the channel to get all the data that is delivered by Cloud.

8. The service processes all the data.
9. The service publishes the data to proxy at Fog level.
10. Edge consumes this data.

It is worth mentioning that each transaction was executed 30 times and tested in 3 environments with the same hardware and software configuration, network conditions.

5 RESULTS AND EVALUATIONS

In this chapter the metrics are arranged a consolidated format by presetting all the experiments throughout this work. While to conducting this study, comparisons had to be made to allow a better understanding of the proposed middleware since they are based on performance validations.

5.1 Experiments

The following scenarios were grounded on a considerable amount of evidences and thus will be discussed in the sections that follow. In addition, the Appendix 5 provides extra information about the results of the experiments.

These result were divided into 4 distinct groups of experiments for a better understating of the middleware performance. Thus, the opportunities to validate the performance of DME despite latency are as follows:

- Experiments without latency and without DME, described in Section 5.3;
- Experiments with latency and without the DME implementation Section 5.4;
- Experiments without latency and with the DME implementation Section 5.5;
- Experiments with latency and with DME implementation Section 5.6.

All four experiments have the same elements as A part of the architectural structure since:

- It operates 2 protocols: CoAP (confirmable) and MQTT (3 QoS);
- Publishes the messages in 3 different payload sizes(small, medium, large);
- Increases user instances to publish messages concurrently.

The difference between the four experiments is narrow in two key areas: the algorithm for DME and latency.

The diagram shown in Figures 5.1, 5.2 provides a detailed view of the planned experiments, as well as the metrics used throughout this work.

Figure 5.1: Test Execution - Architectural components

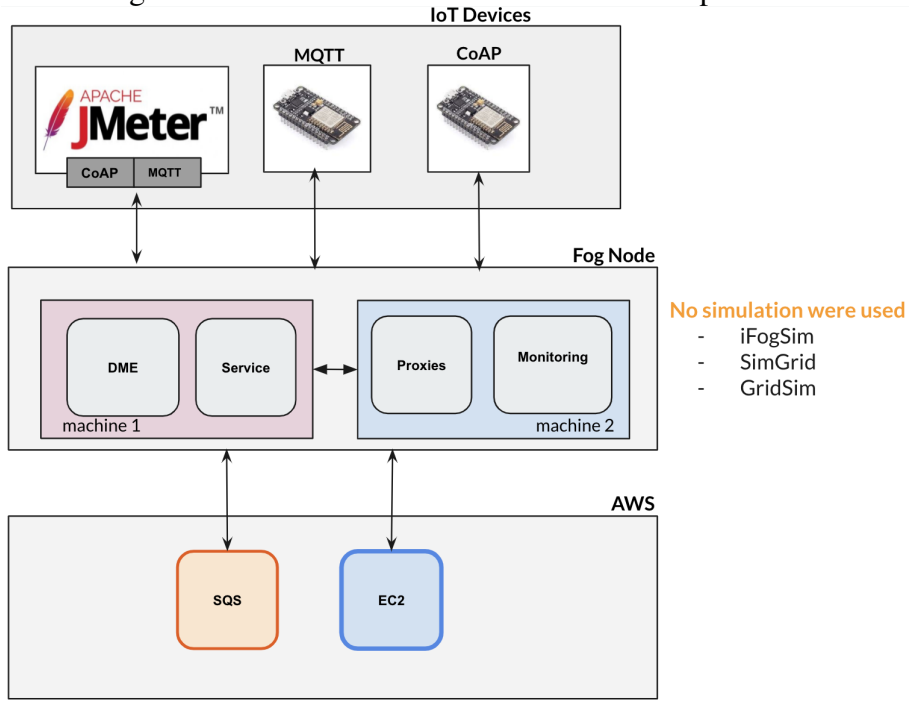
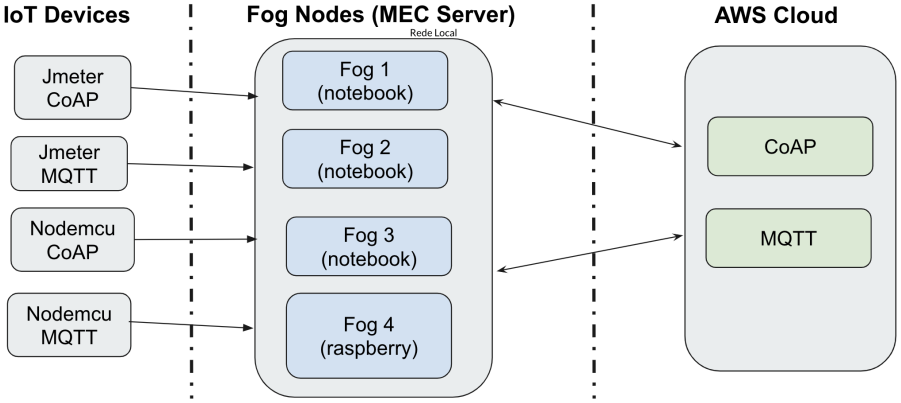


Figure 5.2: MiddleFog Testing Execution (3 Layers: Edge-Fog-Cloud)



5.2 Metrics

This section sets out the metrics that were used throughout this work and also gives a detailed view of the planned experiments.

Metrics are quantifiable measures used to analyze the outcome of a specific process, action or strategy. Performance evaluation (or measurement) is an inseparable process for cross-layers optimization (Cloud, fog, Edge). It highlights weaknesses and benefits by determining whether the technique is fulfilling its objectives. The most common supported metrics for all landscapes have been found to be: resource utilization, energy, response time and latency [6]. Tables 5.1, 5.2 below display the metrics that were used in this work, which are collated and described to results.

Table 5.1: Performance metrics

Metrics
Latency
Response Time

Table 5.2: Variables and measurements for MiddleFog performance assessment

Measurement inputs	Values
Package Message Size	20B, 800B, 2048B
Latency	5% - 30%
Time per seconds	1s , 5s, 10s
CoAP QoS	Confirmable
MQTT QoS	0, 1, 2

5.3 Group I - Experiment with No Latency and No DME

The initial experiment used two protocols, namely CoAP and MQTT, which were based on service quality (CoAP with confirmable QoS and MQTT with QoS 0,1,2), without taking account of latency and without selecting the optimal protocol, and thus demonstrating that there is no DME implementation.

After several executions, Table 5.3 followed the design architecture represented in Chapter 4.1, and the results of the DME performance model were consolidated after several executions. The main factors that validate the experiment, described in Chapter 4.1, are number of users, and package size.

Table 5.3: DME - Consolidated Performance without DME and Latency

Package Size	SMALL			MEDIUM			LARGE		
Number of Users	10	100	200	10	100	200	10	100	200
Protocol									
COAP	0.372	0.345	17.69	0.3777	0.3516	0.333	20.351	17.69	17.693
MQTT QoS 0	20.34	23.83	29.93	20.34	23.83	29.93	21.09	20.14	25.01
MQTT QoS 1	28.35	27.57	27.70	28.87	27.30	26.50	22.89	21.67	25.01
MQTT QoS 2	19.63	19.18	19.79	20.25	19.52	25.13	19.63	19.18	19.79

The observations revealed that:

- Despite the increase in the number of user requests from 10 to 100 and later 200, the CoAP protocol, small and medium- sized packages achieved better results than MQTT, even if there was a change in Quality of Service (0,1,2).
- Although the number of user requests increased from 10 to 100 and later 200,

MQTT Qos 2, large-sized packages achieved better results than MQTT, even after the Quality of Service (0,1) was altered.

5.4 Group II - Experiments with Latency and No DME

The second experiment was conducted with the two CoAP and MQTT protocols based on quality of service (CoAP with QoS confirmed, MQTT with QoS 0,1,2) applied latency, and without choosing the best protocol, and did not have a DME-sign implementation. The table 5.13 shown here, summarizes the results obtained from the DME performance model, based on a series of executions that were conducted in accordance with the design architecture displayed in Chapter 4.1. The primary components that are essential for validating the experiment, as outlined in Chapter 4.1, are the protocol, number of users quantity, and package size.

Table 5.4: DME - Consolidated Performance without DME and applying Latency

Package Size	**SMALL**			**MEDIUM**			**LARGE**		
Number of Users	**10**	**100**	**200**	**10**	**100**	**200**	**10**	**100**	**200**
Protocol									
COAP	0.450	0.435	0.690	0.489	0.492	0.499	22.65	22.89	22.33
MQTT QoS 0	24.34	27.96	29.35	24.44	25.09	29.14	34.13	20.14	25.01
MQTT QoS 1	32.35	32.57	33.05	34.84	34.89	34.93	37.12	37.56	37.91
MQTT QoS 2	23.18	23.52	24.09	23.97	24.56	26.89	24.01	24.55	24.96

The observations revealed that: the protocol selection depends on the application requirements. If the primary concern is latency, the CoAP protocol can be used for small-sized packaged messages sizes.

The code optimization can have a significant impact on system latency. By enhancing the code, it is feasible to reduce the execution time and improve the application performance. Nonetheless, as indicated in Table 5.13, performance is not analyzed on the basis of code optimization, but solely on latency. If the main concern is low latency, the CoAP protocol can be used. The Table below makes a comparison between latency values under optimized code conditions.

Protocol	Size	Good Performance	Worse Performance
Small	**COAP**	0.450	0.3777
Medium	**CoAP**	20.34	20.34
Large	**MQTT QoS 2**	28.35	28.87

5.5 Group III - Experiment No latency and with DME

The third experiment was conducted with the two protocols (CoAP and MQTT) based on quality of service (CoAP with QoS confirmed, MQTT with QoS 0,1,2) and choosing the best protocol, and did not have a DME-sign implementation.

Table 5.14 summarizes the DME performance model results after several executions and follows the design architecture represented in Chapter 4.1. The main features for validate an experiment, mentioned in Chapter 4.1, are: protocol, number of users quantity, and package size.

Table 5.5: DME - Consolidated Performance without latency and with the DME

Package Size	**SMALL**			**MEDIUM**			**LARGE**		
Number of Users	**10**	**100**	**200**	**10**	**100**	**200**	**10**	**100**	**200**
Protocol									
COAP	0.184	0.186	0.199	0.208	0.212	0.250	10.10	10.39	10.86
MQTT QoS 0	12.36	12.62	12.75	12.01	12.66	12.95	13.32	13.34	13.96
MQTT QoS 1	14.12	14.52	14.96	19.75	19.63	19.90	15.20	16.63	17.20
MQTT QoS 2	11.63	11.23	11.83	13.56	13.75	14.53	18.78	18.25	18.85

The observations revealed that the CoAP protocol displays a lower latency than MQTT, and thereby reduces the time taken for CoAP to respond to a request. Nonetheless, MQTT has a much smaller size than CoAP. It should be noted that the size of the code is contingent on upon the number of code lines and the number of dependencies. The smaller the code size, the faster it runs, and the less memory it consumes.

In the table, it can be seen that the CoAP protocol has the lowest latency values of all the sizes and numbers of users. The CoAP protocol has a latency of between 0.184 and 10.86, depending on the size and number of users. The latency of the MQTT protocol depends on the quality of service. The MQTT protocol with QoS 0 shows the lower latency values than QoS 1 and QoS 2. Depending on the size and number of users, the latency of MQTT with QoS 0 ranges from 12.36 to 13.96. However, MQTT with QoS

1 and QoS 2 has higher latency values than QoS 0. MQTT with QoS 1 has a latency of 14.12 to 19.96, while MQTT with QoS 2 has a latency of 11.23 to 18.85, depending on the size and number of users.

5.6 Group IV - Experiment with Latency and DME

The first experiment was conducted with the two protocols, CoAP and MQTT, based on quality of service (CoAP with QoS confirmed, MQTT with QoS 0,1,2) and the best protocol, was chosen that represents the DME-sign implementation.

Table 5.15 consolidates the results of the DME performance model after several executions and follows the designed architecture represented in Chapter 4.1. The main features for validating the experiment, mentioned in Chapter 4.1, are: protocol, number of users, and package size.

Table 5.6: DME - Consolidated Performance with DME and Latency

Package Size	**SMALL**			**MEDIUM**			**LARGE**		
Number of User	**10**	**100**	**200**	**10**	**100**	**200**	**10**	**100**	**200**
Protocol									
COAP	0.192	0.195	0.298	0.252	0.259	0.270	10.731	10.83	10.90
MQTT QoS 0	13.96	13.46	13.42	13.54	14.24	13.42	14.76	14.42	17.34
MQTT QoS 1	16.42	16.52	16.78	21.63	21.17	20.30	16.90	17.00	18.01
MQTT QoS 2	12.80	12.19	12.99	14.56	14.98	15.50	19.63	19.18	19.79

The observations revealed that: there are three different sizes, with corresponding numbers of users. There are two different protocols, CoAP and MQTT, with different quality of service levels. It should be noted that the latency values for the CoAP protocol are relatively low, ranging from 0.192 to 0.298 seconds. This suggests that the CoAP protocol is efficient with regard to latency. In the case of the MQTT protocol with QoS levels 0 and 2, the latency values range from 12.19 to 21.63 seconds. This suggests that the MQTT protocol with these QoS levels, may not be optimized for low latency. The CoAP protocol may be a more suitable option for optimizing a code with low latency requirements.

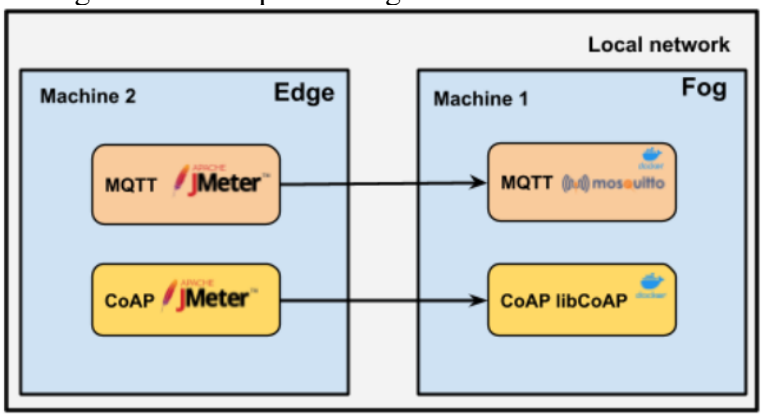
5.7 Final Results

An examination of the packet behavior was conducted in a network generated by the CoAP and MQTT protocols. This analysis seeks to establish parameters that will provide information about which protocol performs best under specific network conditions. The analysis was based on related work that identified key factors in communication performance such as packet size, number of packets generated, and total transmission time.

Initially, a load test was conducted to simulate communication between an Edge device and Fog by making use of the CoAP and MQTT protocols. The load test involved sending messages of different sizes to the Fog. It was configured to generate a new transaction every second and simulate 10 user requests. Subsequently, the packets that were transmitted over the network were scrutinized with the aid of Wireshark, as illustrated in Figures 5.5 and 5.4. The data from the tool's filters and resources was subsequently extracted. The collected data was then used to generate graphs using the R language.

This analysis seeks to provide information about the packet transmission behavior when the CoAP and MQTT protocols are used. Load testing and packet analysis tools provide a comprehensive understanding of communication performance, and lead to the optimization of network protocols and the enhancement of overall efficiency. Figure 5.3 shows the configuration of the test plan created in Jmeter. This configuration is used in both the CoAP and MQTT scripts.

Figure 5.3: Test plan configuration created in Jmeter



Different users requested different transactions, with a new transaction being generated every second. Table 5.7 provides a comprehensive description of the type of transaction and the size of the messages.

CoAP is a protocol designed for use in restricted devices, and is often used in the context of the Internet of Things. One of the main features of CoAP is its support for

Figure 5.4: MQTT wireshark

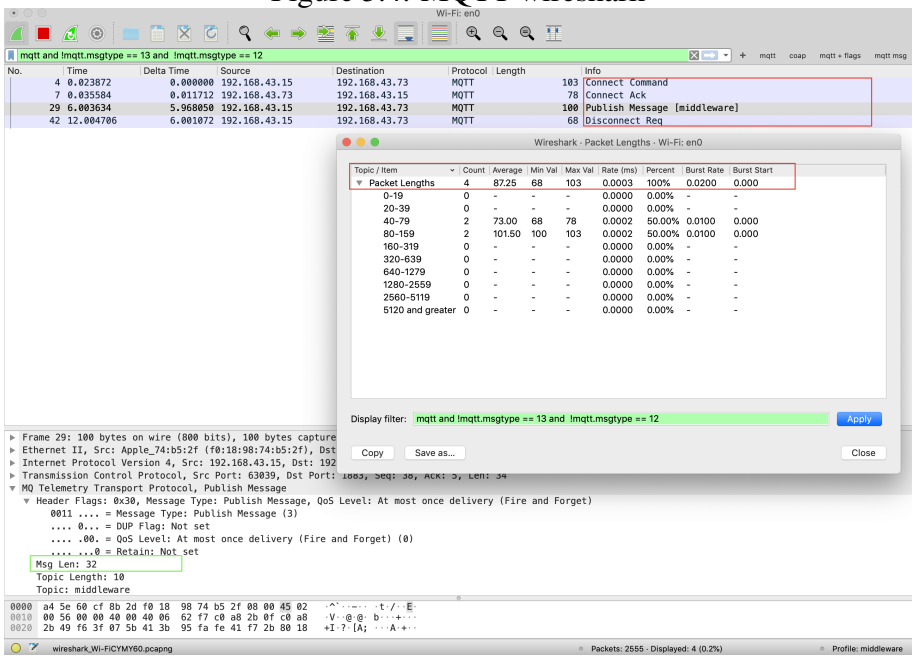


Figure 5.5: CoAP wireshark

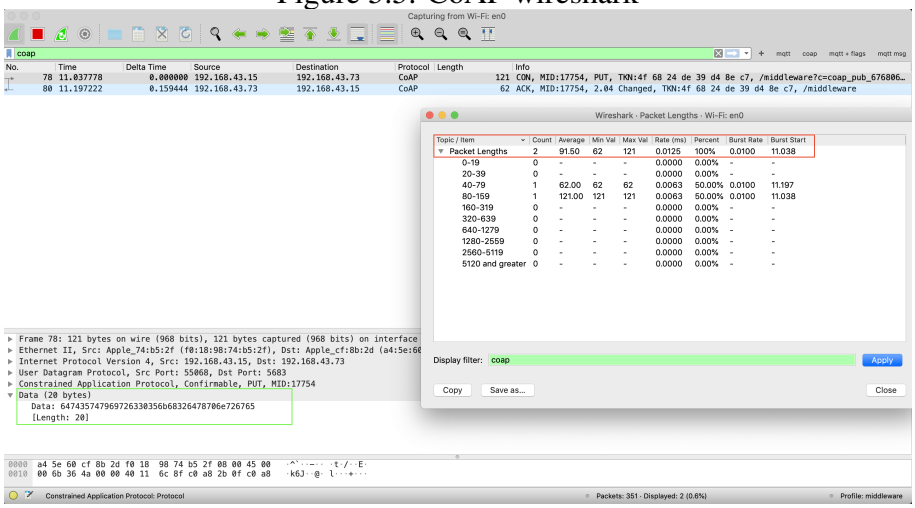


Table 5.7: Characterization of CoAP transactions

CoAP	
Type of Transaction	CON(confirmable)
Message Size	20B, 800B, 2048B

confirmable transactions, which allow reliable communication over untrusted networks. The Table 5.7 provides the results of a load test carried out by means of CoAP, together transactions of type CON and message sizes of 20B, 800B, and 2048B. The objective of this examination was to scrutinize the behavior of packets that were transmitted on the network, and originated from the CoAP protocol. In addition, it sought to, establish the parameters for obtaining data about the optimal performance of the protocol in accordance with network conditions.

On the basis of the data displayed, it can be seen that for all three message sizes, the number of packets generated to transmit all the data is higher for CoAP with transaction type CON than to CoAP with transaction type NON. This can be expected because confirmable transactions require confirmation from the receiving device, which will result in the generation of additional packets.

With regard to the total duration of the data transmission, depending on may vary based on the size of the message. For messages with a size of 20B, the total duration is comparable to both CoAP with CON transaction and CoAP with NON.

In the case of messages exceeding 800B and 2048B, the total duration for CoAP with confirmable transactions is significantly longer. It is likely that additional packets were generated during these transactions. Although confirmable transactions can guarantee dependable communication, they can also lead to an increase in packet generation and prolonged transmission times for larger messages.

The CoAP protocol test was conducted on a total number of 30 occasions, with each session using a different message size and ensuring that the transaction type was verified. MQTT is a lightweight messaging protocol that is widely used in the Internet of Things to communicate between devices. It is designed to be easy and efficient, with a low code volume and low network bandwidth usage. An overview of MQTT and how it is used in IoT applications will be provided in this study. The system employs a publish-subscribe model, in which devices can publish messages on a specific topic and other devices can subscribe to that topic to receive the messages. This ensures efficient communication between devices, as messages are only sent to those that are interested in them. MQTT also provides support for quality of service levels, which provide a means of regulating the reliability of message delivery.

There are three levels of quality of service in the MQTT: QoS 0, QoS 1, and QoS 2. The Quality of Service 0 (QoS 0) is regarded as the least reliable option, as messages are only delivered only once, with the possibility of duplication. In conclusion, the Quality of Service 2 option is the most reliable, as it guarantees the timely delivery of messages, albeit with some potential delays.

MQTT supports different message sizes, from small messages of 20 bytes to larger ones of 800 bytes or even 2,048 bytes. Nonetheless, it should be recognized that the transmission of larger messages may result in a greater use of network bandwidth and a longer duration of transmission. MQTT is a widely utilized protocol for IoT applications because of its simplicity and efficacy, as well as its ability to accommodate diverse levels

of service quality.

Moreover, MQTT can able to be used in a variety of situations, from simple sensor networks to complex industrial systems. Table 5.8 below shows the MQTT of the test.

Table 5.8: Test payload for the MQTT protocol

MQTT	
Quality of Service(QoS)	0,1 and 2
Message Size	20B, 800B, 2048B

The test load for the CoAP protocol was performed 30 times for each quality. The service was undertaken with a different message size to assist understanding. The performance was found to be excellent since. The MQTT protocol is characterized by a the crucial factor Quality of Service (QoS). The guarantee level for message delivery between the sender and the recipient defines the guarantee level for message delivery between the sender and the recipient. The receiver of QoS levels range from zero to 2, where QoS 0 provides the lowest message delivery. Quality of Service 2 provides the highest guarantee level, while QoS 1 provides the lowest guarantee level.

At QoS level 0, messages are delivered without any guarantee, as there is no confirmation from the recipient. Hence, messages with Quality of Service Zero (QoS 0) are ideally suited for messages with low priority that do not delivery guarantees. In our analysis, it was noted that QoS 0 performed well for messages smaller than 20B, since there was minimal overhead and latency in the message delivery. However, in the case of larger message sizes of 800B and 2048B, the performance of QoS 0 was sub-optimal as there was no guarantee of message delivery and messages could be lost in transit.

At quality of service level 1, messages are delivered at least once as the receiver sends an acknowledgment to the sender upon receipt of the message. If the sender does not receive a confirmation, it will resend the message. We found that QoS 1 was reliable for message delivery across message sizes of 20B, 800B, and 2048B. However, there was an increase in latency and message overhead because of the need for acknowledgments.

At QoS level 2, messages are delivered exactly once, as there is a two-way handshake between the sender and receiver. The Quality of Service 2 (QoS 2) provided the most reliable message delivery across all the message sizes. Nonetheless, there was a significant increase in message overhead and latency owing to the need for an additional handshake. The Table 5.9 below, shows the outcomes in accordance with the dimensions of each package.

Table 5.9 displays the packet sizes in bytes for various message sizes, MQTT

Table 5.9: Packet size (bytes)

Protocols	20B	800B	2048B
CoAP	121	481	616
MQTT - QoS 0	100	881	681
MQTT - QoS 1	90	480	380
MQTT - QoS 2	81	277	227

Quality of Service (QoS) levels, and CoAP confirmable transaction type (CON) The term "packet size" refers to the size of the message transmitted in addition to its header. In the case of message size 20B, the packet sizes for MQTT with QoS 0, QoS 1, and QoS 2 were 100, 90, and 81 bytes, respectively. Nevertheless, the packet size for CoAP with CON was 121 bytes. For message size 800B, the packet sizes for MQTT with QoS 0, QoS 1, and QoS 2 were 881, 480, and 277 bytes, respectively. The packet size for COAP with CON was 481 bytes. For message size 2048B, the packet sizes for MQTT with Quality of Service 0, Quality of Service 1, and Quality of Service 2 were 681, 380, and 227 bytes, respectively. The packet size for COAP with CON was 616 bytes. It can be seen from the Table, that the MQTT packet size, rises significantly with the increasing size of the message, particularly with QoS 0. In contrast, the CoAP packet size with CON remains relatively constant in all the message sizes. Table 5.10 shows the results according to the number of packages generated in the table.

Table 5.10: Number of packages generated

Protocols	20B	800B	2048B
CoAP-CON	20	40	74
MQTT - QoS 0	10	10	10
MQTT - QoS 1	20	20	20
MQTT - QoS 2	43	40	40

It should be noted that for message size 20B in Table 5.10, all the QoS levels of MQTT resulted in the generation of 10 packets, whereas CoAP with CON generated 20 packets. All the QoS levels of MQTT generated 20 packets, while CoAP with CON generated 40 with regard to the message size of 2048B, both MQTT QoS 1 and QoS 2, as well as CoAP with CON, generated a total of 74 packets, whereas MQTT QoS 0 generated a total of 43 packets. The number of packets generated increases with the growing size of the message and Quality of Service level. MQTT QoS 0 generates the fewest packets, while CoAP with CON generates the most packets. The following Table 5.11 shows the total transmission time.

The total period required to transmit data of varying sizes through diverse protocols. The protocols compared in the table are MQTT with Quality of Service 0, QoS 1

Table 5.11: Total data transmission time (values in seconds)

Protocols	20B	800B	2048B
CoAP-CON	6.9	7.13	7.1
MQTT - QoS 0	12.7	12.8	12.8
MQTT - QoS 1	12.4	12.8	12.8
MQTT - QoS 2	12.9	12.7	12.6

and QoS 2, and CoAP with CON transaction type.

The findings indicate that CoAP achieves a better performance than MQTT in terms of the total duration required to transmit data. For instance, in a message size of 20B, CoAP (together with transaction type CON) needed a timeframe of 6.9 seconds to transmit data, whereas MQTT with QoS 0, QoS 1, and QoS 2 had times of 12.7, 12.4, and 12.9 seconds, respectively. This suggests that CoAP is capable of transmitting smaller messages in a shorter duration than MQTT.

However, when it comes to transmitting larger messages, MQTT performs better than CoAP. For instance, in a message size of 800B, MQTT with Quality of Service 0, Quality of Service 1, and Quality of Service 2 required a period of 12.8 seconds to transmit data, whereas CoAP with the CON transaction type required a period of 7.13 seconds. This suggests that MQTT can transmit large messages more efficiently than CoAP.

The performance of MQTT is contingent on the QoS level employed. The time taken to transmit data is lower when there is an improvement in quality of service. For example, for a message size of 20B, MQTT with QoS 2 took a shorter time to transmit data than QoS 0 and QoS 1. This is because QoS 2 ensures that the message is delivered to the receiver.

This study has attempted to compare the performance of CoAP and MQTT protocols based on parameters related to the size and number of packets, as well as the total time required to transmit the data. According to the research findings, the protocol that performed best varied, depending on the size of the packets transmitted.

In the case of small packets of 20B, it was noted that the CoAP protocol was more efficient in terms of speed, despite the larger number of packets that had to be transmitted through the network. However, when it comes to larger packets, the MQTT protocol achieved a better performance. Furthermore, tests were conducted with average packet sizes of 800 bytes, and it was found that the MQTT protocol with QoS level 2 had a better performance than the MQTT protocol with QoS level 1.

On the basis of the results obtained from this study, we recommend specific con-

figurations for each packet size. In the case of small 20B packets, it is recommended that CoAP is used with CON. In the case of medium packets (800B), MQTT is recommended with QoS level 2. However, in cases where the priority is higher in medium packets, it is advisable to use MQTT with QoS Level 1. The study underlines the importance of selecting protocol settings according to packet size to achieve optimal performance. The ensuing Table 5.12 shows the performance of the CoAP and MQTT protocols at various Quality of Service (QoS) levels for varying message sizes and user counts.

Table 5.12: CoAP and MQTT protocol performance with different QoS levels

Package Size	**SMALL**			**MEDIUM**			**LARGE**		
User Quantity	**10**	**100**	**200**	**10**	**100**	**200**	**10**	**100**	**200**
Protocol									
COAP	0.372	0.345	17.69	0.3777	0.3516	0.333	20.351	17.69	17.693
MQTT QoS 0	20.34	23.83	29.93	20.34	23.83	29.93	21.09	20.14	25.01
MQTT QoS 1	28.35	27.57	27.70	28.87	27.30	26.50	22.89	21.67	25.01
MQTT QoS 2	19.63	19.18	19.79	20.25	19.52	25.13	19.63	19.18	19.79

Different protocols, namely CoAP and MQTT, were compared with regard to their performance, as well as in terms of size and number of users. The code size is divided into small, medium, and large, while the number of users is divided into 10, 20, and 100.

The CoAP protocol displays a lower latency than MQTT, an thus results in a shorter response time needed by CoAP to respond to a request. However, MQTT has a smaller size than CoAP. It should be noted that the code size determined by the number of code lines and the number of dependencies. The lower the code size, the faster it will run and the less memory it will consume.

In Table 5.12, it can be seen that the CoAP protocol shows the lowest latency values for all sizes and numbers of users. The latency of the CoAP protocol varies from 0.184 to 10.86, contingent upon the size and number of users. The latency of the MQTT protocol depends on the quality of service.

The MQTT protocol with QoS 0 has lower latency values than QoS 1 and QoS 2. The MQTT protocol has lower latency values than QoS 1 and QoS 2. MQTT with QoS 0 varies from 12.36 to 13.96, depending on the size and number of users. However, MQTT, with QoS 1 and QoS 2, has higher latency values than QoS 0. MQTT with QoS 1 ranges from 14.12 to 19.96, while the latency of MQTT with QoS 2 ranges from 11.23 to 18.85, depending on the size and number of users.

The protocol selection depends on the application requirements, as described in the DME algorithm(Figure 4.8). If the main concern is low latency, the CoAP protocol can be used. Nonetheless, in the event of the code size being significant significance, the MQTT protocol with QoS 0 can be used.

Table 5.13: DME - Consolidated Performance without DME and applying Latency

Package Size	**SMALL**			**MEDIUM**			**LARGE**		
Number of Users	**10**	**100**	**200**	**10**	**100**	**200**	**10**	**100**	**200**
Protocol									
COAP	0.450	0.435	0.690	0.489	0.492	0.499	22.65	22.89	22.33
MQTT QoS 0	24.34	27.96	29.35	24.44	25.09	29.14	34.13	20.14	25.01
MQTT QoS 1	32.35	32.57	33.05	34.84	34.89	34.93	37.12	37.56	37.91
MQTT QoS 2	23.18	23.52	24.09	23.97	24.56	26.89	24.01	24.55	24.96

The code size has been classified into small, medium, and large, while the number of users has been classified into 10, 100, and 200. The CoAP protocol shows a lower latency than the MQTT protocol in a wide range of sizes numbers of users. The CoAP protocol latency varies between 0.450 and 0.690, depending on the size and number of users. The latency of the MQTT protocol depends on the quality of service.

The MQTT protocol with QoS 0 has higher latency values than QoS 1 and QoS 2. The MQTT protocol has higher latency values than QoS 1 and QoS 2. The latency of MQTT with Quality of Service 0 ranges from 24.34 to 34.13, depending on the size and number of users. Nevertheless, MQTT with QoS 1 and QoS 2 has lower latency values than QoS 0. The latency of MQTT with QoS 1 varies from 32.35 to 37.91, while the latency of MQTT with QoS 2 varies from 23.18 to 24.96, depending on the size and number of users.

The optimization of the code has the potential to significantly impact the latency of a system. By optimizing the code, it is possible to improve the execution time and application performance. Nonetheless, it should be noted that the table displayed does not analyze performance on the basis of code optimization, but solely on latency. If the main concern is low latency, the CoAP protocol can be used. Table 5.14 gives a comparison of latency values under optimized code conditions.

The code size is categorized into small, medium, and large, while the user quantity is categorized into 10, 100, and 200. The latency values are lower than the MQTT protocols in all the sizes and numbers of users. The CoAP latency varies between 0.450

Table 5.14: DME - Consolidated Performance without latency and with the DME

Package Size	SMALL			MEDIUM			LARGE		
Number of Users	10	100	200	10	100	200	10	100	200
Protocol									
COAP	0.184	0.186	0.199	0.208	0.212	0.250	10.10	10.39	10.86
MQTT QoS 0	12.36	12.62	12.75	12.01	12.66	12.95	13.32	13.34	13.96
MQTT QoS 1	14.12	14.52	14.96	19.75	19.63	19.90	15.20	16.63	17.20
MQTT QoS 2	11.63	11.23	11.83	13.56	13.75	14.53	18.78	18.25	18.85

and 0.690, depending on the size and number of users. In contrast, the latency of MQTT with QoS 0 is higher than that of CoAP, yet still lower than other MQTT QoS levels. Depending on the size and number of users, the latency of MQTT with QoS 0 varies from 13.96 to 17.34.

However, as one moves to higher MQTT QoS levels, the latency values will increase significantly. If low latency is a concern, CoAP or MQTT with QoS 0 can be employed. If the primary concern is the low latency, CoAP or MQTT with QoS 0 may be utilized . However, if reliability and delivery guarantee are more important, MQTT with higher QoS levels can be considered. The following Table 5.14 shows the level of quality of services provided.

Table 5.15: DME - Consolidated Performance with DME and Latency

Package Size	SMALL			MEDIUM			LARGE		
Number of Users	10	100	200	10	100	200	10	100	200
Protocol									
COAP	0.192	0.195	0.298	0.252	0.259	0.270	10.731	10.83	10.90
MQTT QoS 0	13.96	13.46	13.42	13.54	14.24	13.42	14.76	14.42	17.34
MQTT QoS 1	16.42	16.52	16.78	21.63	21.17	20.30	16.90	17.00	18.01
MQTT QoS 2	12.80	12.19	12.99	14.56	14.98	15.50	19.63	19.18	19.79

There exist three distinct sizes, namely small, medium, and large, each of which caters for a specific number of users. In the case of each size and number of users, there are latency values for two different protocols, CoAP and MQTT, with different levels of quality of service. With regard to the CoAP protocol, it can be seen that the latency values range from 0.192 to 0.98 seconds. This suggests that the CoAP protocol is efficient

in terms of latency. In contrast, the latency values for the MQTT protocol with QoS levels 0 and 2, the latency values are relatively high, ranging from 12.19 to 21.63 seconds. This suggests that the MQTT protocol with these Quality of Service levels may not have been optimized for low latency. CoAP may be a better option for optimizing a code with low latency requirements. Table 5.16 shows the quality analysis without taking account of latency and optimization.

Table 5.16: Latency-free quality and optimization

Package Size	**SMALL**			**MEDIUM**			**LARGE**		
Number of Users	**10**	**100**	**200**	**10**	**100**	**200**	**10**	**100**	**200**
Protocol									
COAP	0.372	0.345	17.69	0.3777	0.3516	0.333	20.351	17.69	17.693
MQTT QoS 0	20.34	23.83	29.93	20.34	23.83	29.93	21.09	20.14	25.01
MQTT QoS 1	28.35	27.57	27.70	28.87	27.30	26.50	22.89	21.67	25.01
MQTT QoS 2	19.63	19.18	19.79	20.25	19.52	25.13	19.63	19.18	19.79

The CoAP protocol has a relatively low latency,and ranges from 0.345 to 0.377 seconds. This indicates that the CoAP protocol is effective for a certain number of users of different sizes i.e. the latency values for the MQTT protocol with QoS level 0, range from 20.14 to 29.93 seconds. The latency values for QoS level 1 are lower, ranging from 21.67 to 28.87 seconds. The latency values for QoS level 2 are relatively consistent, ranging from 19.18 to 25.13 seconds.

In light of this, it may be better to use the CoAP protocol to reduce latency in a given context. It should be noted consider that the choice of protocol and QoS level must be based on specific system requirements and trade-offs between latency and other factors, such as reliability and network bandwidth.

On the basis of findings, it may be feasible to select a protocol and quality of service level that are optimized for low latency and efficient performance in a particular setting. If low latency is a critical requirement, the CoAP protocol may be a suitable option, since it has demonstrated relatively low latency values. This feature could be advantageous for applications that require instantaneous communication, such as video conferencing, online gaming, and remote control systems.

CoAP was designed to facilitate effective communication between devices that possess limited processing power and memory, while also reducing the communication costs of in low-power and lossy networks. In terms of cost, the CoAP protocol is designed

to be effective and reduce the need for network resources. The binary format employed in this application reduces the size of messages and effectively implements compression to reduce of the size of data payloads. This will reduce bandwidth usage and network overhead, which can be converted into lower communication costs. This product is specifically designed for use with devices that possess limited processing power and memory, which implies that it can require inexpensive hardware, and thus make a communications solution affordable. Unlike other protocols, such as MQTT, which may need additional processing power and memory, (and thus incur higher communication costs), the CoAP protocol has been specifically designed to be lightweight and effective.

[76] The demand for effective and Internet-compatible solutions in IoT networks was analyzed and, as a result, there is an increasing use of sensors and devices that are interconnected via the Internet. The author discusses the limitations of many IoT devices, such as power, processing, and limitations of memory capacity. These limitations create a demand for protocols can improve communication without overloading the devices. CoAP and DTLS are two protocols specifically designed for constrained devices that provide both enforcement and security protocols.

[76] designed a prototype to solve an agricultural problem involving sensors and an IoT device. The article provides a practical IoT solution involving CoAP and DTLS. The results suggest that CoAP was efficient with regard to transmission quality and latency, which makes it suitable for technological solutions in the agriculture.

[3] Two specific protocols, MQTT and CoAP, were examined, which are used to ensure efficiency the management of network traffic in the Internet of Things (IoT). IoT has become very popular in recent years and has enabled devices to connect with other in new ways. The authors conducted a series of experiments to contrast the effectiveness of MQTT and CoAP protocols. The results demonstrated that MQTT was more accurate in ensuring packet delivery than CoAP. Nonetheless, CoAP had a superior performance when transmitting a limited number of messages.

[55] The importance of maintaining the confidentiality of sensitive information in the context of the Internet of Things is discussed. The paper evaluates various symmetrical encryption mechanisms to guarantee confidentiality for messages sent through the MQTT and CoAP protocol. The author analyzed the performance of different cryptographic algorithms by means of metrics such as power consumption and response time. The study shows that choosing a cryptographic algorithm can lead to significant energy savings of up to 32.29% and a 41.60% in CPU usage. The findings suggest that the selection of the

cryptographic algorithm must take into account the availability of device resources.

[75] We compared the communication delay and network traffic between the two most common protocols, Constrained Application Protocol (COAP) and Message Queuing Telemetry Transport (MQTT), by creating two systems. On average, COAP was found to have smaller packet sizes and no keep-alive messages. The system is well-crafted to allow interaction with systems that operate on Hypertext Transfer Protocol (HTTP) However, MQTT has a lower communication delay and is simpler to implement. It is designed to be capable of automatically forwarding messages to multiple clients without an additional configuration. The Quality of Service (QoS) is incorporated into the MQTT design to ensure the delivery of messages. The authors conclude that the selection of protocol ought to be based on the specific requirements of the application. If the system requires lightweight communication with small packet sizes and no keep-alive messages, COAP is a suitable choice. However, if there is a need for a low communication delay and the ability to automatically forward messages to multiple clients, MQTT would be a better option.

[4] investigated two specific protocols, MQTT and CoAP, which are used to ensure the efficiency and management of communication traffic in the Internet of Things. The Internet of Things (IoT) has become increasingly popular in recent years, and enabled devices to communicate with each other in novel ways. The authors carried out a set of experiments to compare the performance of MQTT and CoAP protocols. The results demonstrated that MQTT was more accurate in guaranteeing packet delivery than CoAP. However, CoAP is superior when sending a limited number of messages.

[54] provides an overview of MQTT and its variations adapted for sensor networks, called MQTT-SN. The authors conducted a comparative analysis of these protocols with other prevalent IoT application layer protocols, such as CoAP. The authors emphasize the advantages and disadvantages of each protocol. The results show that MQTT and MQTT-SN have several advantages over CoAP, including more efficient communication, lower energy consumption, and better scalability. In the view of the authors, CoAP still raises questions and challenges in this area, including security, interoperability, and scalability.

[11] compared the performance of CoAP, MQTT-SN and HTTP protocols in scenarios that simulate smart homes to determine which are the most energy efficient and have the lowest rate of packet loss and latency. The author created and ran scenarios to obtain power consumption metrics, latency, and packet loss data. After analyzing the

data, it was found that the CoAP protocol had a better performance than MQTT-SN and HTTP in all the graphs, which corroborates the findings results of this study.

[43] underlines the importance of CoAP in attributing efficiency network resources and reducing their use. The lightweight design and use of a binary format have the potential to reduce network resources and implement compression to reduce of the data payload size, and thus cut communication costs. The type of resource is particularly useful in countries and regions where Internet access is limited, as it provides a quality solution at a low cost. [43] concluded that the CoAP protocol is a promising solution for the Web of Things initiative, since it is designed to be efficient and suitable for devices with limited resources. By bringing CoAP support to the web browser, it will be easier to develop applications for small and resource-constrained devices. This will encourage the adoption of digital technologies and improve the overall user experience in the IoT industry.

[14] The challenge of effectively facilitating machine-to-machine communication in the Internet of Things (IoT) was analyzed in the context of limited bandwidth, unreliability, and intermittent wireless communication links. The authors compare the performance of four IoT protocols, namely MQTT, CoAP, DDS, and a custom protocol based on UDP, in a medical environment. The protocols were evaluated by means of a network emulator that tested a wireless access network with low bandwidth, high system latency, and high packet loss. DDS results in higher bandwidth usage than MQTT, but its superior performance regarding latency and data reliability makes it an attractive choice for IoT medical applications and more. This suggests that the appropriate protocol should be chosen on the basis of the specific requirements of the application.

[36] made a comparison between the Internet of Things (IoT) protocols that are for data transfer in constrained IoT networks. The authors recognize that it can be a challenging task to create a network with many interconnected physical IoT devices. One of the main challenges in the IoT world is how to efficiently support machine-to-machine communication in constrained networks. To address this issue, the paper conducts an evaluation of the performance and contrasts two commonly employed protocols, namely Message Queuing Telemetry Transport (MQTT) and Constrained Application Protocol (CoAP) When developing IoT applications, it can be difficult to determine which protocol to use, as there are several factors to that have to be taken into account.

As a means of to contrasting the effectiveness of MQTT and CoAP, Hedi and [36] conducted experiments in different scenarios and assessed the effectiveness of the two protocols on the basis of various parameters, including message delivery rate, mes-

sage delivery time, and consumption power. The findings of the study revealed that both MQTT and CoAP demonstrated a superior performance in various scenarios, and the selection of either protocol would be contingent on the specific requirements of the IoT application. The study revealed that MQTT had a better performance in scenarios where the primary focus was on reliability and message delivery rate, whereas CoAP proved to be more suitable for scenarios that required low latency and efficient communication, in accordance with the findings of this study.

[71] The Internet of Things, which is based on IPv6 (6IoT), needs End-to-End security (E2E) because of the sensitive nature of potential applications and the presence of humans in the loop. The authors decided to implement Secure CoAPs (CoAPs) to provide End-to-End security in 6IoT systems that are based on CoAPs.

Smartphones that possess sensing capabilities, Internet connectivity, and processing and storage capabilities are a crucial component of 6IoT. In the paper, the authors design, implement, and evaluate CoAPs for Android smartphones, which they refer to as INDIGO. This is the first attempt to provide support for CoAPs on smartphones. The authors implement and evaluate all the cryptographic cipher suites recommended in the CoAP protocol, including certificate-based authentication using ecliptic curve cryptography (ECC).

[23] believe that the development of a CoAP extension will support real-time communication between sensors, actuators, and users in IoT applications. The authors made modifications to the Message Options field of the CoAP header to prevent messages from being sent unless they could be checked within set deadlines, thus reducing network traffic. This extension can be implemented on low-cost smartphones that can be purchased for less than $100, which makes it a viable solution for developing countries with vulnerable communities.

[23] propose a scenario for the implementation of an efficient public transport system in developing nations, where informal transport vehicles are provided with identification marks and empowered to plan their routes and fares upon request. This extension of CoAP will enable real-time updates on transport availability to a specific destination, and improve the overall performance of the open network. This application scenario is having a significant impact on the standard of living of many people who must travel across large cities in developing nations, and it plays a significant role in bringing about what is often referred to as the social good. The authors argue that traffic quality of service is one of the main factors related to the Internet of Things in developing countries

with limited bandwidth. The planned CoAP extension could be used control message transmission through the worst-case transmission delay, and avoid transmission in cases where deadlines are not met. This will greatly improve the overall performance of the open network.

The potential of using simple protocols to handle time-constrained IoT traffic is a notable cost-effective means of improving the quality of life in developing countries with poor digitization of services. The planned CoAP extension is easily implemented and offers a viable solution to improve real-time communication in IoT applications.

CoAP [15] is an effective application protocol in an IoT environment. The authors explain that IoT is a concept that combines the physical and digital worlds, by allowing devices connected to the Internet to observe and act. These observations can then be linked to cloud-based digital services, and allow intelligent decision-making which will result in a significant uptake because of to their potential benefits.

Nonetheless, the expansion of the Internet of Things is still constrained by a number of factors, including the efficiency of network and edge sensing devices, the creation of appropriate applications, and security concerns. To address these concerns, the authors conducted experiments to validate the effectiveness of CoAP as a transport protocol in a low-power personal area network.

The experiments were centered on assessing communication in an environment with limited connectivity. The results that CoAP was an efficient transport protocol in low signal strength environments. This study provides valuable information regarding the effectiveness of CoAP in an IoT environment, which can assist in developing suitable applications to ensure secure and reliable communication.

The environment of Web-based Internet of Things is a part of Mobility management [29]. As the number of sensors and embedded devices increases, CoAP has emerged as a widely utilized protocol. However, it is essential to support mobility management in an environment like this. The authors plan a set of mobility management protocols based on CoAP, referred to as CoMP-G, which is an extension of the existing CoMP protocol. In the proposed scheme, one of the body sensors serves as a coordinator and relays all the control messages to the web-of-things mobility management system (WMMS) on behalf of the other body sensors. Each WMMS also stores information from the body sensor array. The authors conduct a numerical analysis and show that the scheme provides a better performance than the existing CoMP protocol in terms of total signaling and delivery delay.

The CoAP-G protocol addresses the limitations of the CoMP protocol, which was designed for the mobility of single sensor nodes and experiences inadequacies in group-based mobility. The CoAP-G protocol is designed to enable effective group-based mobility management in a web-based IoT environment. This constitutes a significant contribution to the field of IoT and has the potential to enhance the efficacy and effectiveness of IoT applications involving group-based mobility. The efficiency protocol has the potential to significantly enhance the management of group-based mobility in a web-based IoT environment, and the numerical analysis conducted by the authors provides evidence of its that its performance is superior to that of the existing CoMP protocol.

According to research findings, it has been determined that the CoAP protocol is a lightweight and effective protocol that was designed to establish communication between devices with limited processing power and memory, and to reduce the communication costs in low-power and lossy networks. CoAP has a big major, namely of its low latency values, which can make it a suitable system for applications that require real-time communication. In contrast with other protocols, such as MQTT, which may need additional processing power and memory, (resulting in higher communication costs), the CoAP protocol has been specifically designed to be lightweight and efficient, which makes it a promising solution for the Web of Things (WoT)

The lightweight design and binary format of this application are crucial in reducing network resources, and thus reducing communication costs. This can be particularly useful in countries and regions where Internet access is limited. Incorporating CoAP support into the web browser makes it easier to develop applications for diminutive and resource-constrained devices, which assist in the adoption of digital technologies and enhance the overall user experience in the IoT industry.

They compared the IoT protocols used for data transfer in restricted IoT networks,[36] because they recognized that one of the main challenges in the IoT world is to support machine-to-machine communication in restricted conditions. The authors emphasize the significance of selecting a protocol and QoS level that are optimized for low latency and efficient performance in a particular context. Furthermore, the findings of their study suggest that CoAP may be a superior alternative to MQTT for reducing latency in certain situations.

In general, the CoAP protocol offers numerous advantages, including low latency values, efficient use of network resources, and suitability for devices with limited processing power and memory. These benefits make it a promising solution for standard-

izing Internet access, particularly in countries and regions where resources are limited. Nonetheless, it should be noted that the selection of the protocol and QoS level ought to be based on established system requirements and the balance between latency and other factors, such as reliability and network bandwidth.

The CoAP solution offers a good prospect for making the Internet of Things (IoT) more accessible in developing nations, particularly for applications that necessitate low latency and efficient communication. It has several features that make it suitable for IoT applications in resource-constrained environments, including low overhead and power consumption, as well as support for unreliable networks. These features allow CoAP to operate effectively on low-quality or limited-access networks.

This protocol can help improve the quality of the Internet in countries with limited or low-quality access because it is designed to work efficiently in on restricted networks. For instance, in nations with a limited network infrastructure and low Internet penetration, CoAP-based protocols can enhance the overall user experience by overcoming the obstacles of limited bandwidth and network congestion. Furthermore, it can help reduce the power consumption of IoT devices, which is especially important in resource-constrained environments where the battery life is limited.

CoAP uses the same underlying architecture as HTTP, which makes it easy to integrate with existing web technologies. This means it could help expand the reach of the Internet to new devices and applications, including those in developing countries with limited or low-quality access.

6 CONCLUSION AND RECOMMENDATIONS FOR FUTURE WORK

This chapter discusses the conclusions of this work and and makes recommendations for future work in which the experiments can be extended.

6.1 Closing Remarks

The Internet of Things enables billions of smart devices to capture, process, and transform data to improve decision making. It requires a critical mobile edge computing ecosystem to establish dependability. It also requires a highly efficient and distributed architecture with multiple IoT communication protocols. Moreover, an intelligent middleware is needed to achieve efficiency, throughput, and reliability in data delivery across diverse protocols without any interference from the local setup of the device.

This work has carried out a study of Middleware Fog as a use case for Protocol Interoperability in heterogeneous Internet of Things environments. The MiddleFog middleware is a modular and interoperable system that selects the most appropriate communication protocol among MQTT and CoAP. It establishes a decision-making heuristic in monitoring network conditions and has the payload size of the message. Monitoring occurs within the communication channel between Edge device-Fog Node and Fog Node-Cloud so that a decision can be made about which protocol will have the best performance. Currently, we are engaged in MQTT and COAP testing.

The MiddleFog is responsible for the following a) fulfilling critical tasks, b) performing high computational loading, c) receiving notification from the Decision-Making Engine about the best protocol in the network, d) introducing protocol exchange, e) optimizing the amount of data per package, and f) making improvements without affecting performance.

The results of the experiments described in Chapter 5. It is evident that MiddleFog effectively mitigates limitations in the area of communication that result from latency, package loss, and inadequate network throughput between MEC and Cloud. Initial evaluations show that the message loss rate is lower than 25% for small messages, and that performance improves by around 48% for medium-sized delivery messages. It should be noted that the experiments were conducted in real scenarios, and all of them were carried out in three different machines and networks to avoid errors. No simulator was used to handle the real problem in software environments.

6.2 Future Work

With regard to the experiments carried out this work, some improvements and adjustments may be needed in the light of further discoveries some of these improvements might include the following factors:

- Since there is middleware that is currently working in the Fog layer, there is an opportunity to create Domain Specific Language (DSL) for Edge. Focusing on this can assist in improving performance in all three layers (Edge-Fog-Cloud);
- The middleware interoperability can be improved by supporting more protocols, such as HTTP 2, AMQP, SMQTT, and DDS. This will increase the middleware interface by adding IoT protocols;
- A decision not to use any type of simulator was made for the Edge or Fog layers, after experiencing several library problems such as a publishing messages above 2K. Thus there is a need to change the library to a more scalable approach;
- There is a need to expand the measurements criteria in terms of performance of the machine when running DME and to ask, what is the level of impact on resources, if the devices are slower.

BIBLIOGRAPHY

[1] Kassem Ahmad et al. “IoT: Architecture, Challenges, and Solutions Using Fog Network and Application Classification”. Em: *2018 International Arab Conference on Information Technology (ACIT)*. Werdanye, Lebanon: IEEE, nov. de 2018, pp. 1–7. ISBN: 9781728103853. DOI: <10.1109/ACIT.2018.8672696>. URL: <https://ieeexplore.ieee.org/document/8672696/> (acesso em 18/09/2020).

[2] Mudassar Ali et al. “Joint Cloudlet Selection and Latency Minimization in Fog Networks”. Em: *IEEE Transactions on Industrial Informatics* 14.9 (set. de 2018), pp. 4055–4063. ISSN: 1551-3203, 1941-0050. DOI: <10.1109/TII.2018.2829751>. URL: <https://ieeexplore.ieee.org/document/8345659/> (acesso em 13/11/2020).

[3] Alyaziya Almheiri e Zakaria Maamar. “IoT Protocols – MQTT versus CoAP”. Em: *Proceedings of the 4th International Conference on Networking, Information Systems & Security*. NISS ’21. KENITRA, AA, Morocco: Association for Computing Machinery, 2021. ISBN: 9781450388719. DOI: <10.1145/3454127.3456594>. URL: <https://doi.org/10.1145/3454127.3456594>.

[4] Alyaziya Almheiri e Zakaria Maamar. “IoT Protocols – MQTT versus CoAP”. Em: *Proceedings of the 4th International Conference on Networking, Information Systems & Security*. NISS ’21. KENITRA, AA, Morocco: Association for Computing Machinery, 2021. ISBN: 9781450388719. DOI: <10.1145/3454127.3456594>. URL: <https://doi.org/10.1145/3454127.3456594>.

[5] Makkad Asim. “A Survey on Application Layer Protocols for Internet of Things (IoT)”. Em: *International Journal of Advanced Research in Computer Science* 8.1 (mar. de 2017). ISSN: 0976-5697. (Acesso em 16/09/2020).

[6] Mohammad S. Aslanpour, Sukhpal Singh Gill e Adel N. Toosi. “Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research”. Em: *Internet of Things* 12 (2020), p. 100273. ISSN: 2542-6605. DOI: <https://doi.org/10.1016/j.iot.2020.100273>. URL: <https://www.sciencedirect.com/science/article/pii/S2542660520301062>.

[7] Kitchenham BA e Stuart Charters. “Guidelines for performing Systematic Literature Reviews in Software Engineering”. Em: 2 (jan. de 2007).

[8] G. Banavar et al. “An efficient multicast protocol for content-based publish-subscribe systems”. Em: *Proceedings. 19th IEEE International Conference on Distributed Computing Systems (Cat. No.99CB37003)*. Austin, TX, USA: IEEE Comput. Soc, 1999, pp. 262–272. ISBN: 9780769502229. DOI: <10.1109/ICDCS.1999.776528>. URL: <http://ieeexplore.ieee.org/document/776528/> (acesso em 13/11/2020).

[9] Sharu Bansal e Dilip Kumar. “IoT Application Layer Protocols: Performance Analysis and Significance in Smart City”. Em: *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. Kanpur, India: IEEE, jul. de 2019, pp. 1–6. ISBN: 9781538659069. DOI: <10.1109/ICCCNT45670.2019.8944807>. URL: <https://ieeexplore.ieee.org/document/8944807/> (acesso em 18/09/2020).

[10] Sharu Bansal e Dilip Kumar. “IoT Ecosystem: A Survey on Devices, Gateways, Operating Systems, Middleware and Communication”. en. Em: *International Journal of Wireless Information Networks* 27.3 (set. de 2020), pp. 340–364. ISSN: 1068-9605, 1572-8129. DOI: <10.1007/s10776-020-00483-7>. URL: <http://link.springer.com/10.1007/s10776-020-00483-7> (acesso em 16/09/2020).

[11] Abimael Silas BARROS. “Análise de desempenho dos protocolos coap, mqtt-sn e http em redes IoT”. Em: Trabalho de Conclusão de Curso (Graduação em Redes de Computadores)- Universidade Federal do Ceará,., 2022.

[12] Lorenzo Bassi. “Industry 4.0: Hope, hype or revolution?” Em: *2017 IEEE 3rd International Forum on Research and Technologies for Society and Industry (RTSI)*. Modena, Italy: IEEE, set. de 2017, pp. 1–6. ISBN: 9781538639061. (Acesso em 27/06/2020).

[13] Gaetano Carlucci, Luca De Cicco e Saverio Mascolo. “HTTP over UDP: an experimental investigation of QUIC”. en. Em: *Proceedings of the 30th Annual ACM Symposium on Applied Computing - SAC ’15*. Salamanca, Spain: ACM Press, 2015, pp. 609–614. ISBN: 9781450331968. DOI: <10.1145/2695664.2695706>. URL: <http://dl.acm.org/citation.cfm?doid=2695664.2695706> (acesso em 22/10/2020).

[14] Yuang Chen e Thomas Kunz. “Performance evaluation of IoT protocols under a constrained wireless access network”. Em: abr. de 2016, pp. 1–7. DOI: <10.1109/MoWNet.2016.7496622>.

[15] Louis Coetzee, Dawid Oosthuizen e Buhle Mkhize. “An Analysis of CoAP as Transport in an Internet of Things Environment”. Em: mai. de 2018.

[16] Walter Colitti et al. “Evaluation of constrained application protocol for wireless sensor networks”. Em: out. de 2011, pp. 1–6. ISBN: 978-1-4577-1264-7. DOI: <10.1109/LANMAN.2011.6076934>.

[17] Burak H. Corak et al. “Comparative Analysis of IoT Communication Protocols”. Em: *2018 International Symposium on Networks, Computers and Communications (ISNCC)*. Rome: IEEE, jun. de 2018, pp. 1–6. ISBN: 9781538637791. DOI: <10.1109/ISNCC.2018.8530963>. URL: <https://ieeexplore.ieee.org/document/8530963/> (acesso em 22/10/2020).

[18] Angelo Corsaro. *The Data Distribution Service Tutorial*. Mai. de 2014.

[19] D. Happ, A. Wolisz. “Limitations of the Pub/Sub Pattern for Cloud Based IoT and Their Implications.” Em: *Cloudification of Internet of Things (CIoT) Paris* (2016), pp. 1–6.

[20] Madhavi Dave, Jyotika Doshi e Harshal Arolkar. “MQTT- CoAP Interconnector: IoT Interoperability Solution for Application Layer Protocols”. Em: *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*. 2020, pp. 122–127. DOI: <10.1109/I-SMAC49090.2020.9243377>.

[21] Biljana Dimitrova e Aleksandra Mileva. “Steganography of Hypertext Transfer Protocol Version 2 (HTTP/2)”. Em: *Journal of Computer and Communications* 05.05 (2017), pp. 98–111. ISSN: 2327-5219, 2327-5227. DOI: <10.4236/jcc.2017.55008>. URL: <http://www.scirp.org/journal/doi.aspx?DOI=10.4236/jcc.2017.55008> (acesso em 22/10/2020).

[22] Jasenka Dizdarević et al. “A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration”. en. Em: *ACM Computing Surveys* 51.6 (fev. de 2019), pp. 1–29. ISSN: 0360-0300, 1557-7341. DOI: <10.1145/3292674>. URL: <https://dl.acm.org/doi/10.1145/3292674> (acesso em 15/09/2020).

[23] Gabriel M. Eggly et al. “Real-Time Primitives for CoAP: Extending the Use of IoT for Time Constraint Applications for Social Good”. Em: *Proceedings* 2.19 (2018). ISSN: 2504-3900. DOI: <10.3390/proceedings2191257>. URL: <https://www.mdpi.com/2504-3900/2/19/1257>.

[24] Joel L. Fernandes et al. “Performance evaluation of RESTful web services and AMQP protocol”. Em: *2013 Fifth International Conference on Ubiquitous and Future Networks (ICUFN)*. ISSN: 2165-8536. Jul. de 2013, pp. 810–815. DOI: <10.1109/ICUFN.2013.6614932>.

[25] Iulia Florea et al. “Survey of Standardized Protocols for the Internet of Things”. Em: *2017 21st International Conference on Control Systems and Computer Science (CSCS)*. Bucharest, Romania: IEEE, mai. de 2017, pp. 190–196. ISBN: 9781538618394. DOI: <10.1109/CSCS.2017.33>. URL: <http://ieeexplore.ieee.org/document/7968561/> (acesso em 16/09/2020).

[26] Ala Al-Fuqaha et al. “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications”. Em: *IEEE Communications Surveys & Tutorials* 17.4 (2015), pp. 2347–2376. ISSN: 1553-877X, 2373-745X. DOI: <10.1109/COMST.2015.2444095>. URL: <https://ieeexplore.ieee.org/document/7123563/> (acesso em 17/09/2020).

[27] Konstantinos Fysarakis et al. “Which IoT Protocol? Comparing Standardized Approaches over a Common M2M Application”. Em: *2016 IEEE Global Communications Conference (GLOBECOM)*. Washington, DC, USA: IEEE, dez. de 2016, pp. 1–7. ISBN: 9781509013289. DOI: <10.1109/GLOCOM.2016.7842383>. URL: <http://ieeexplore.ieee.org/document/7842383/> (acesso em 17/09/2020).

[28] Antonio Carlos Gil. *Métodos e técnicas de pesquisa social*. 6. ed. Ediitora Atlas SA, 2008.

[29] Moneeb Gohar, Jin-Ghoo Choi e Seok-Joo Koh. “CoAP-based group mobility management protocol for the Internet-of-Things in WBAN environment”. Em: *Future Generation Computer Systems* 88 (jun. de 2018). DOI: <10.1016/j.future.2018.06.003>.

[30] Jayavardhana Gubbi et al. “Internet of Things (IoT): A vision, architectural elements, and future directions”. Em: *Future Generation Computer Systems* 29.7 (2013). Including Special sections: Cyber-enabled Distributed Computing for Ubiquitous Cloud and Network Services & Cloud Computing and Scientific Applications — Big Data, Scalable Analytics, and Beyond, pp. 1645–1660. ISSN: 0167-739X. DOI: <https://doi.org/10.1016/j.future.2013.01.010>. URL: <http://www.sciencedirect.com/science/article/pii/S0167739X13000241>.

[31] Jasmin Guth et al. “A Detailed Analysis of IoT Platform Architectures: Concepts, Similarities, and Differences”. Em: *Internet of Everything*. Ed. por Beniamino Di Martino et al. Singapore: Springer Singapore, 2018, pp. 81–101. ISBN: 9789811058608 9789811058615. DOI: <10.1007/978-981-10-5861-5_4>. URL: <http://link.springer.com/10.1007/978-981-10-5861-5_4> (acesso em 16/09/2020).

[32] Guth, J., Breitenbücher, U., Falkenthal, M., Fremantle, P., Kopp, O., Leymann, F., Reinfurt, L. “A detailed analysis of IoT platform architectures: concepts, similarities, and differences.” Em: *Springer - Internet of Everything Singapore* 26.5 (2018), pp. 81–101.

[33] Guth, J., U. Breitenbucher, M. Falkenthal, F. Leymann, and L. Reinfurt. “Comparison of IoT platform architectures: A field study based on a reference architecture.” Em: *Cloudification of the Internet of Things (CIoT) IEEE* (2019).

[34] Guth, J., U. Breitenbucher, M. Falkenthal, F. Leymann, and L. Reinfurt. “Comparison of IoT platform architectures: A field study based on a reference architecture.” Em: *Cloudification of the Internet of Things (CIoT) IEEE* (2019).

[35] David Hanes et al. *IoT fundamentals: networking technologies, protocols, and use cases for the Internet of things*. Cisco Press fundamentals series. OCLC: ocn929582969. Indianapolis, Indiana, USA: Cisco Press, 2017. ISBN: 9781587144561.

[36] I. Heđi, I. Špeh e A. Šarabok. “IoT network protocols comparison for the purpose of IoT constrained networks”. Em: *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2017, pp. 501–505. DOI: <10.23919/MIPRO.2017.7973477>.

[37] Elis Hernandes et al. “Using GQM and TAM to evaluate StArt - a tool that supports Systematic Review”. en. Em: *CLEI Electronic Journal* 15 (abr. de 2012), pp. 3–3. ISSN: 0717-5000. URL: <http://www.scielo.edu.uy/scielo.php?script=sci_arttext&pid=S0717-50002012000100003&nrm=iso>.

[38] Annika Hinze, Kai Sachs e Alejandro Buchmann. “Event-based applications and enabling technologies”. en. Em: *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems - DEBS ’09*. Nashville, Tennessee: ACM Press, 2009, p. 1. ISBN: 9781605586656. DOI: <10.1145/1619258.1619260>. URL: <http://portal.acm.org/citation.cfm?doid=1619258.1619260> (acesso em 13/11/2020).

[39] Juan F. Ingles-Romero et al. “A Model-Driven Approach to Enable Adaptive QoS in DDS-Based Middleware”. Em: *IEEE Transactions on Emerging Topics in Computational Intelligence* 1.3 (jun. de 2017), pp. 176–187. ISSN: 2471-285X. DOI: <10.1109/TETCI.2017.2669187>. URL: <http://ieeexplore.ieee.org/document/7857781/> (acesso em 22/10/2020).

[40] Syed Rameez Ullah Kakakhel et al. “A Qualitative Comparison Model for Application Layer IoT Protocols”. Em: *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*. Rome, Italy: IEEE, jun. de 2019, pp. 210–215. ISBN: 9781728117966. DOI: <10.1109/FMEC.2019.8795324>. URL: <https://ieeexplore.ieee.org/document/8795324/> (acesso em 08/11/2020).

[41] Hasnain Kashif, Muhammad Nasir Khan e Qasim Awais. “Selection of Network Protocols for Internet of Things Applications: A Review”. Em: *2020 IEEE 14th International Conference on Semantic Computing (ICSC)*. San Diego, CA, USA: IEEE, fev. de 2020, pp. 359–362. ISBN: 9781728163321. DOI: <10.1109/ICSC.2020.00072>. URL: <https://ieeexplore.ieee.org/document/9031509/> (acesso em 17/09/2020).

[42] Barbara Kitchenham. “Procedures for Performing Systematic Reviews”. Em: (2004).

[43] Matthias Kovatsch. “CoAP for the Web of Things: From Tiny Resource-Constrained Devices to the Web Browser”. Em: *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*. UbiComp ’13 Adjunct. Zurich, Switzerland: Association for Computing Machinery, 2013, pp. 1495–1504. ISBN: 9781450322157. DOI: <10.1145/2494091.2497583>. URL: <https://doi.org/10.1145/2494091.2497583>.

[44] In Lee e Kyoochun Lee. “The Internet of Things (IoT): Applications, investments, and challenges for enterprises”. en. Em: *Business Horizons* 58.4 (jul. de 2015), pp. 431–440. ISSN: 00076813. DOI: <10.1016/j.bushor.2015.03.008>. URL: <https://linkinghub.elsevier.com/retrieve/pii/S0007681315000373> (acesso em 17/09/2020).

[45] Olivier Levillain. “Parsifal: A Pragmatic Solution to the Binary Parsing Problems”. Em: *2014 IEEE Security and Privacy Workshops*. San Jose, CA: IEEE, mai. de 2014, pp. 191–197. ISBN: 9781479951031. DOI: <10.1109/SPW.2014.35>. URL: <http://ieeexplore.ieee.org/document/6957303/> (acesso em 14/01/2022).

[46] Miguel Angel Lopez Pena e Isabel Munoz Fernandez. “SAT-IoT: An Architectural Model for a High-Performance Fog/Edge/Cloud IoT Platform”. Em: *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. Limerick, Ireland: IEEE, abr. de 2019, pp. 633–638. ISBN: 9781538649800. DOI: <10.1109/WF-IoT.2019.8767282>. URL: <https://ieeexplore.ieee.org/document/8767282/> (acesso em 13/11/2020).

[47] Andriy Luntovskyy e Larysa Globa. “Performance, Reliability and Scalability for IoT”. Em: *2019 International Conference on Information and Digital Technologies (IDT)*. Zilina, Slovakia: IEEE, jun. de 2019, pp. 316–321. ISBN: 9781728114019. DOI: <10.1109/DT.2019.8813679>. URL: <https://ieeexplore.ieee.org/document/8813679/> (acesso em 12/11/2020).

[48] Pavel Masek et al. “Implementation of True IoT Vision: Survey on Enabling Protocols and Hands-On Experience”. en. Em: *International Journal of Distributed Sensor Networks* 12.4 (abr. de 2016), p. 8160282. ISSN: 1550-1477, 1550-1477. DOI: <10.1155/2016/8160282>. URL: <http://journals.sagepub.com/doi/10.1155/2016/8160282> (acesso em 18/09/2020).

[49] Roberto Morabito et al. “Consolidate IoT Edge Computing with Lightweight Virtualization”. Em: *IEEE Network* 32.1 (jan. de 2018), pp. 102–111. ISSN: 0890-8044, 1558-156X. DOI: <10.1109/MNET.2018.1700175>. URL: <https://ieeexplore.ieee.org/document/8270640/> (acesso em 16/09/2020).

[50] MQTT. *MQTT MQTT Specification*. 2020. URL: <https://mqtt.org/mqtt-specification> (acesso em 18/09/2020).

[51] *OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0, Part 0: Overview*. URL: <http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-overview-v1.0-os.html> (acesso em 18/09/2020).

[52] OASIS OPEN. *MQTT Version 3.1.1*. 2020. URL: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html> (acesso em 18/09/2020).

[53] Laboratorio de Pesquisa Em Engenharia de Software. *StArt*. 2022. URL: <http://lapes.dc.ufscar.br/tools/start_tool> (acesso em 20/08/2020).

[54] Silvio Quincozes, Tubino Emilio e Juliano Kazienko. “MQTT Protocol: Fundamentals, Tools and Future Directions”. Em: *IEEE Latin America Transactions* 17 (set. de 2019), pp. 1439–1448. DOI: <10.1109/TLA.2019.8931137>.

[55] Vagner Quincozes, Silvio Quincozes e Juliano Kazienko. “Avaliando a Sobrecarga de Mecanismos Criptográficos Simétricos na Internet das Coisas: Uma Comparação Quantitativa entre os Protocolos MQTT e CoAP”. Em: *Anais do XX Workshop em Desempenho de Sistemas Computacionais e de Comunicação*. Evento Online: SBC, 2021, pp. 13–24. DOI: <10.5753/wperformance.2021.15719>. URL: <https://sol.sbc.org.br/index.php/wperformance/article/view/15719>.

[56] RFC. *The Constrained Application Protocol (CoAP)*. 2020. URL: <http://www.rfc-editor.org/rfc/rfc7252.txt> (acesso em 18/09/2020).

[57] S. N. Gaikwad S. D. Shewale. “An IoT based real-time weather monitoring system using Raspberry Pi.[]” Em: *International Journal of Advanced Research in Electrical - Electronics and Instrumentation Engineering* 6.6 (2018), pp. 4242–4249.

[58] Cleber Santana, Brenno Alencar e Cassio Prazeres. “Microservices: A Mapping Study for Internet of Things Solutions”. Em: *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*. Cambridge, MA: IEEE, nov. de 2018, pp. 1–4. ISBN: 9781538676592. DOI: <10.1109/NCA.2018.8548331>. URL: <https://ieeexplore.ieee.org/document/8548331/> (acesso em 13/11/2020).

[59] Shashank Shekhar et al. “URMILA: A Performance and Mobility-Aware Fog/Edge Resource Management Middleware”. Em: *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*. Valencia, Spain: IEEE, mai. de 2019, pp. 118–125. ISBN: 9781728101514. DOI: <10.1109/ISORC.2019.00033>. URL: <https://ieeexplore.ieee.org/document/8759356/> (acesso em 12/11/2020).

[60] Gabriel Signoretti et al. “Performance Evaluation of an Edge OBD-II Device for Industry 4.0”. Em: *2019 II Workshop on Metrology for Industry 4.0 and IoT (MetroInd4.0&IoT)*. Naples, Italy: IEEE, jun. de 2019, pp. 432–437. ISBN: 9781728104294. (Acesso em 27/06/2020).

[61] Manisha Singh e Gaurav Baranwal. “Quality of Service (QoS) in Internet of Things”. Em: *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*. Bhimtal: IEEE, fev. de 2018, pp. 1–6. ISBN: 9781509067855. DOI: <10.1109/IoT-SIU.2018.8519862>. URL: <https://ieeexplore.ieee.org/document/8519862/> (acesso em 08/11/2020).

[62] Inés Sittón-Candanedo et al. “Edge Computing Architectures in Industry 4.0: A General Survey and Comparison”. Em: jan. de 2020, pp. 121–131. ISBN: 978-3-658-07615-3. DOI: <10.1007/978-3-030-20055-8_12>.

[63] Mariusz Slabicki e Krzysztof Grochla. “Performance evaluation of CoAP, SNMP and NETCONF protocols in fog computing architecture”. Em: *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*. Istanbul, Turkey: IEEE, abr. de 2016, pp. 1315–1319. ISBN: 9781509002238. DOI: <10.1109/NOMS.2016.7503010>. URL: <http://ieeexplore.ieee.org/document/7503010/> (acesso em 13/11/2020).

[64] Soma Bandyopadhyay et al. “Role Of Middleware For Internet Of Things: A Study”. Em: *International Journal of Computer Science & Engineering Survey* 2.3 (ago. de 2011), pp. 94–105. ISSN: 09763252. DOI: <10.5121/ijcses.2011.2307>. URL: <http://www.airccse.org/journal/ijcses/papers/0811cses07.pdf> (acesso em 16/09/2020).

[65] S Sreeraj e G Santhosh Kumar. “Performance of IoT protocols under constrained network, a Use Case based approach”. Em: *2018 International Conference on Communication, Computing and Internet of Things (IC3IoT)*. Chennai, India: IEEE, fev. de 2018, pp. 495–498. ISBN: 9781538624593. DOI: <10.1109/IC3IoT.2018.8668105>. URL: <https://ieeexplore.ieee.org/document/8668105/> (acesso em 23/01/2022).

[66] Daniel Stenberg. “HTTP2 explained”. en. Em: *ACM SIGCOMM Computer Communication Review* 44.3 (jul. de 2014), pp. 120–128. ISSN: 0146-4833. DOI: <10.1145/2656877.2656896>. URL: <https://dl.acm.org/doi/10.1145/2656877.2656896> (acesso em 22/10/2020).

[67] Jose Paolo Talusan et al. “Evaluating Performance of In-Situ Distributed Processing on IoT Devices by Developing a Workspace Context Recognition Service”. Em: *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. Kyoto, Japan: IEEE, mar. de 2019, pp. 633–638. ISBN: 9781538691519. DOI: <10.1109/PERCOMW.2019.8730693>. URL: <https://ieeexplore.ieee.org/document/8730693/> (acesso em 13/11/2020).

[68] Andrew S. Tanenbaum e David Wetherall. *Computer Networks*. 5ª ed. Boston: Prentice Hall, 2011. ISBN: 978-0-13-212695-3. URL: <https://www.safaribooksonline.com/library/view/computer-networks-fifth/9780133485936/>.

[69] Dinesh Thangavel et al. “Performance evaluation of MQTT and CoAP via a common middleware”. Em: *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. Singapore: IEEE,

abr. de 2014, pp. 1–6. ISBN: 9781479928439 9781479928422. DOI: <10.1109/ISSNIP.2014.6827678>. URL: <http://ieeexplore.ieee.org/document/6827678/> (acesso em 23/01/2022).

[70] *The Function of Quality Assurance (QA) with the Internet of Things - Hybrid Cloud and IT Solutions*. en-US. Jun. de 2019. URL: <https://www.ctl.io/blog/post/qa-with-the-iot> (acesso em 27/06/2020).

[71] Daniele Trabalza, Shahid Raza e Thiemo Voigt. "INDIGO: Secure CoAP for Smartphones". Em: *Wireless Sensor Networks for Developing Countries*. Ed. por Faisal Karim Shaikh et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 108–119. ISBN: 978-3-642-41054-3.

[72] Hoa Tran-Dang e Dong-Seong Kim. "Fog Computing: Fundamental Concepts and Recent Advances in Architectures and Technologies". Em: *Cooperative and Distributed Intelligent Computation in Fog Computing: Concepts, Architectures, and Frameworks*. Cham: Springer Nature Switzerland, 2023, pp. 1–18. ISBN: 978-3-031-33920-2. DOI: <10.1007/978-3-031-33920-2_1>. URL: <https://doi.org/10.1007/978-3-031-33920-2_1>.

[73] Namaste UI. "How Important is Quality Assurance for the Internet of Things IoT". en-US. Em: *namasteui.com* 54 (). ISSN: 01403664. DOI: <10.1016/j.comcom.2014.09.008>. URL: <https://www.namasteui.com/how-important-is-quality-assurance-for-the-internet-of-things-iot> (acesso em 16/01/2022).

[74] Luis M. Vaquero et al. *A break in the clouds: towards a cloud definition*. Dez. de 2009. URL: <https://doi.org/10.1145/1496091.1496100> (acesso em 27/06/2020).

[75] Henri W. van der Westhuizen e Gerhard Petrus Hancke. "Practical Comparison between COAP and MQTT - Sensor to Server level". Em: *2018 Wireless Advanced (WiAd)* (2018), pp. 1–6. URL: <https://api.semanticscholar.org/CorpusID:57190743>.

[76] Johann Westphall. "Desenvolvimento de uma Aplicação com dispositivo IoT usando Protocolos DTLS e CoAP". Em: TCC(graduação) - Universidade Federal de Santa Catarina. Centro Tecnológico. Ciências da Computação., 2018.

[77] *What is DDS?* URL: <https://www.dds-foundation.org/what-is-dds-3/> (acesso em 22/10/2020).

[78] Thomas Wiss e Stefan Forsstrom. “Feasibility and performance evaluation of SCTP for the industrial internet of things”. Em: *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*. Beijing: IEEE, out. de 2017, pp. 6101–6106. ISBN: 9781538611272. DOI: <10.1109/IECON.2017.8217060>. URL: <http://ieeexplore.ieee.org/document/8217060/> (acesso em 13/11/2020).

APPENDIX A — GRANULAR FLOW EXPERIMENT

This chapter provides additional information about the methods employed in the experiments that were conducted. It includes extra details addressed to a single Group I. Part of subsection 5.3. The sequence of experiments without DME and latency is shown. Furthermore, a detailed overview is given of the planned experiments and the metrics used throughout this work.

When carrying out this experimental study, it was essential to make comparisons. These provide a more comprehensive understanding of middleware types of based on performance validations. The subsequent scenarios have provided a good deal of evidence with success by assessing the system performance on the basis of two key elements: DME and latency. The visualization includes scenarios based on package size and, number of users.

A.1 Experiments No Latency and No DME

The experiments using all the protocols were based on quality of service(CoAP with confirmable QoS , MQTT with Qos 0,1,2) without any latency and without choosing the best protocol;in addition, it represents no DME implementation.

A.1.1 Group I - CoAP Small

This scenario evaluates the total time in seconds for CoAP transactions with a small-sized package (20B), without any latency and in a situation where DME does not give information about the protocol. Each payload size was executed 30 times following the test plan outlined in Table 5.2.

The goal of the first experiment is to obtain the total response time as a metric for CoAP transactions with small-sized messages(20B), as shown in Table A.1 and Graphic A.1. The experiment is divided in terms of the number of users (10, 100, 200) with the configuration described in Table 5.2. The results are based on the numbers of users: 10 users are shown in Graphic A.2, 100 users in users in Graphic A.3, 200 users in Graphic A.4.

Table A.1 consolidates the total time in seconds for the three layers Edge, Fog and

Table A.1: CoAP Small - Total Time in Seconds

CoAP No Latency No DME	Average Total Time in Seconds
	Small (20B)
10 users	0.3726
100 users	0.3452
200 users	17.69

Cloud. The following graphs show the data and presented the transactions by time per second. Each bar represents a transaction and the transactions with the same time. When the time is equal, the transactions are displayed above the other, as shown in Graphs A.3 and A.4.

Figure A.1: CoAP Small - Total time includes the number of users

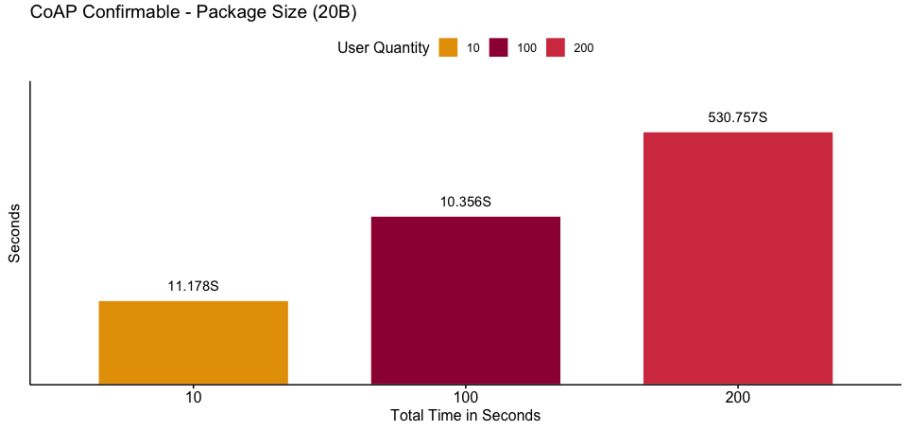


Figure A.2: CoAP Small - Transactions 10 users

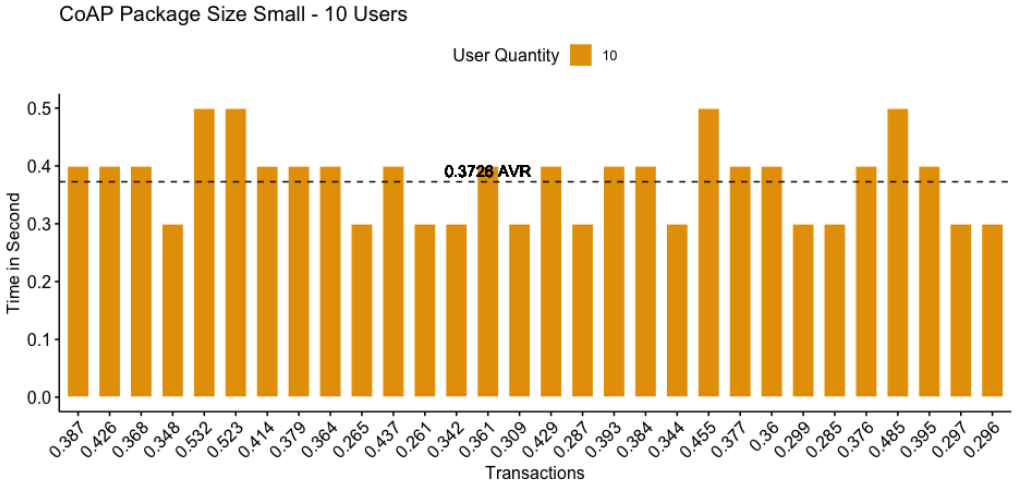


Figure A.3: CoAP Small - Transactions 100 users

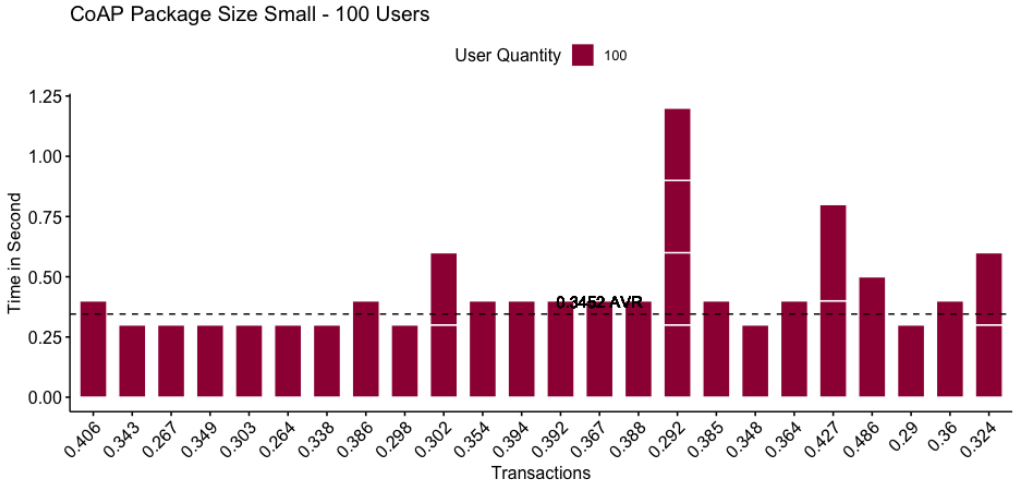
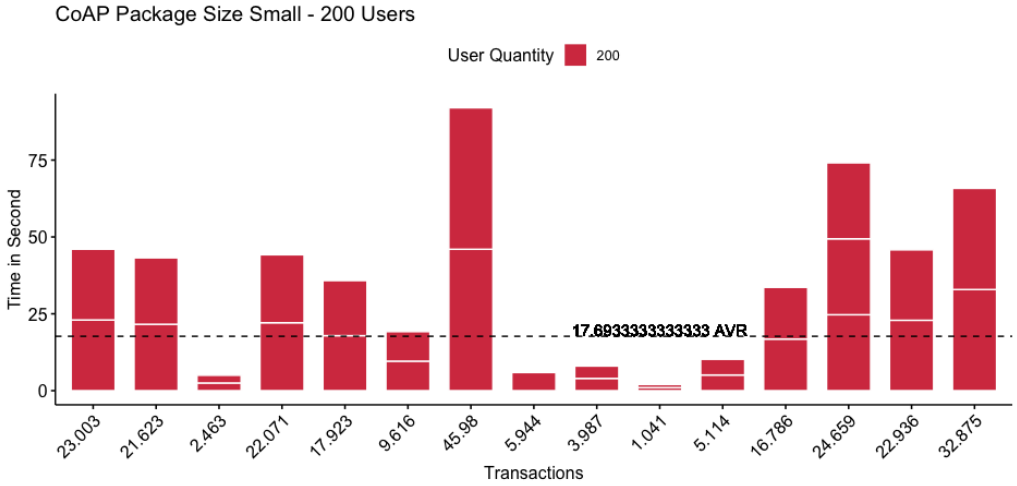


Figure A.4: CoAP Small - Transactions 200 users



A.1.2 Group I - CoAP Medium

This scenario evaluates the total time in seconds for CoAP transactions with medium-sized packages (800B), without any latency and the DME does not give information about the protocol. Each payload size was executed 30 times following the test plan outlined in Table 5.2.

The purpose of the first experiment is to obtain the total response time as a metric for CoAP transactions with small-size messages (800B) as shown in Table A.2 and Graph A.5. The experiment is divided into numbers of users (10, 100, 200) with the configuration described in Table 5.2. The results are based on the number of users: 10 users are shown Graph A.6, 100 users Graph A.7 , 200 users in Graph A.8.

Table A.2 consolidates the total time in seconds three layers Edge, Fog and Cloud. The following graps for the data show the transactions by time per second, each bar rep-

Table A.2: CoAP Medium - Total Time in Seconds

CoAP No Latency No DME	Average Total Time in Seconds
	Medium (800B)
10 users	0.3777
100 users	0.3516
20 users	0.3533

resents a transaction and the transactions with the same time. When the time is equal, the transactions are displayed in above the other, as shown in Graphs A.7 and A.8.

Figure A.5: CoAP Medium - Total time includes the number of users

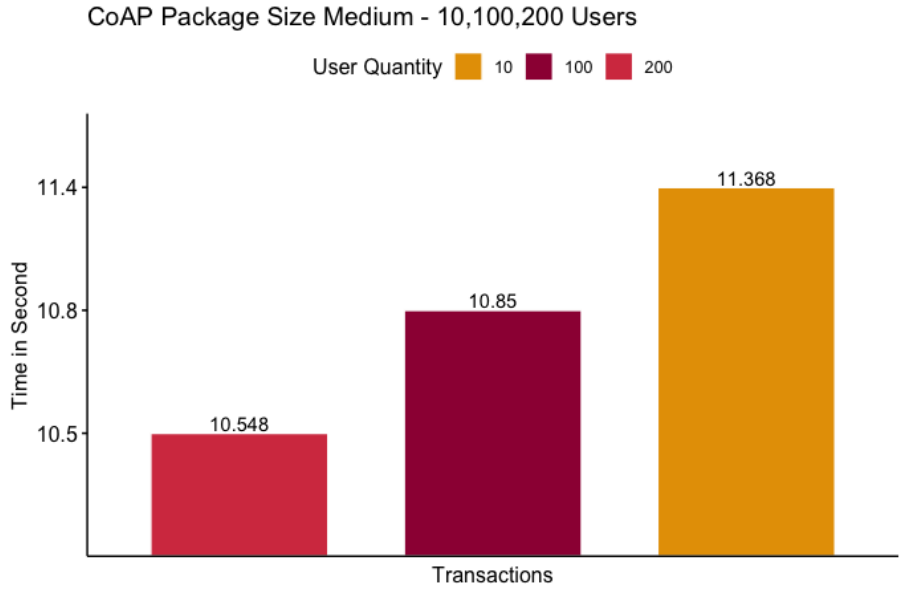


Figure A.6: CoAP Medium - Transactions 10 users

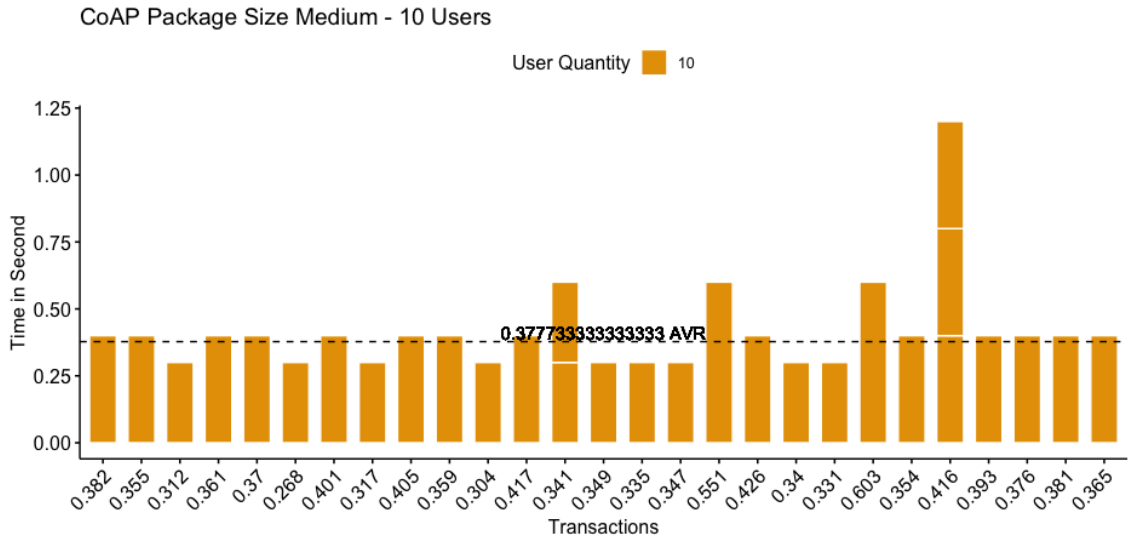


Figure A.7: CoAP Medium - Transactions 100 users

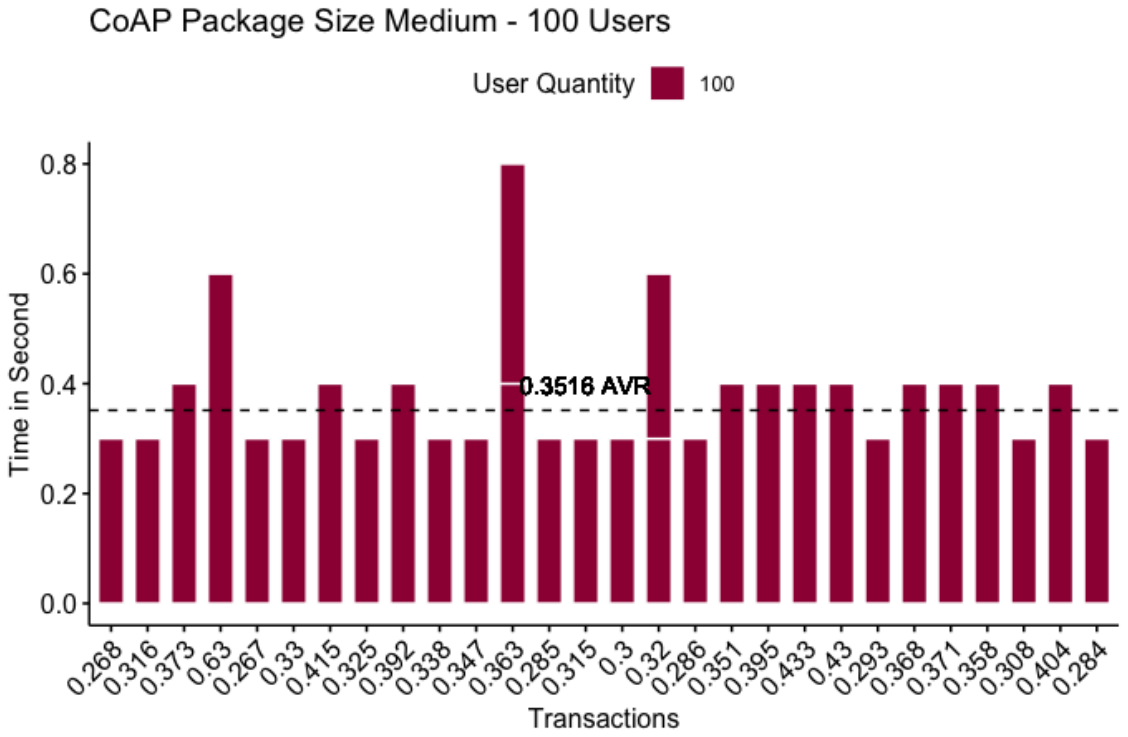
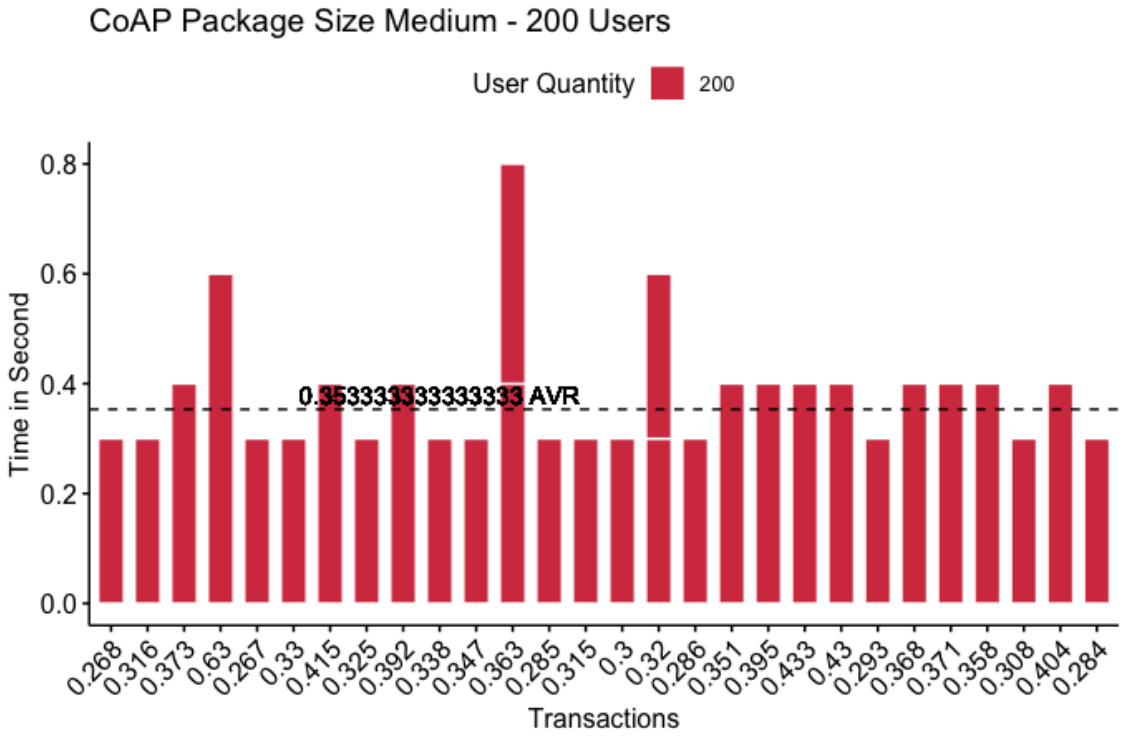


Figure A.8: CoAP Medium - Transactions 200 users



A.1.3 Group I - CoAP Large

This scenario evaluates the total time in seconds for CoAP transactions with large-sized packages (2048B), without any latency and where DME does not give any information about the protocol. Each payload size was executed 30 times following the test plan outlined in Table 5.2.

The purpose of the first experiment is to obtain the total response time as a metric for CoAP transactions with small-size messages (2048B), as shown in Table A.3 and Graph A.9. The experiment is divided into number of users (10, 100, 200) with the configuration described in Table 5.2. The results are based on the number of users: 10 users are shown in Graph A.10, 100 users in Graph A.11, 200 users in Graph A.12.

Table A.3: CoAP Large - Total Time in Seconds

CoAP No Latency No DME	**Average Total Time in Seconds**
	Large (2048B)
10 users	20.351
100 users	17.6919
200 users	17.693

Table A.3 consolidates the total time in seconds displayed by three layers, Edge, Fog and Cloud. The following graphs for the data show the transactions by time per second each bar represents a transaction and the transactions with the same time. When the time is equal, the transactions are displayed above the other, as shown in Graphs A.11 and A.12.

Figure A.9: CoAP Large - Total time includes all the users

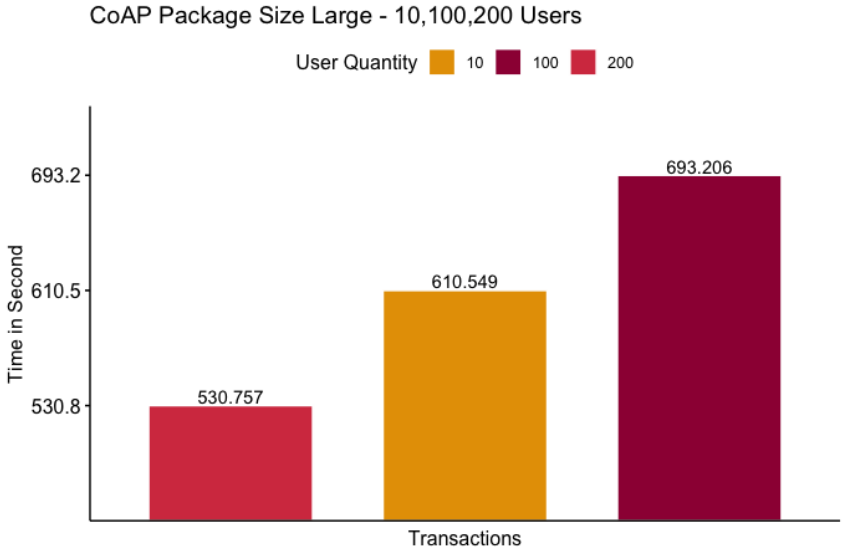


Figure A.10: CoAP Large - Transactions 10 users

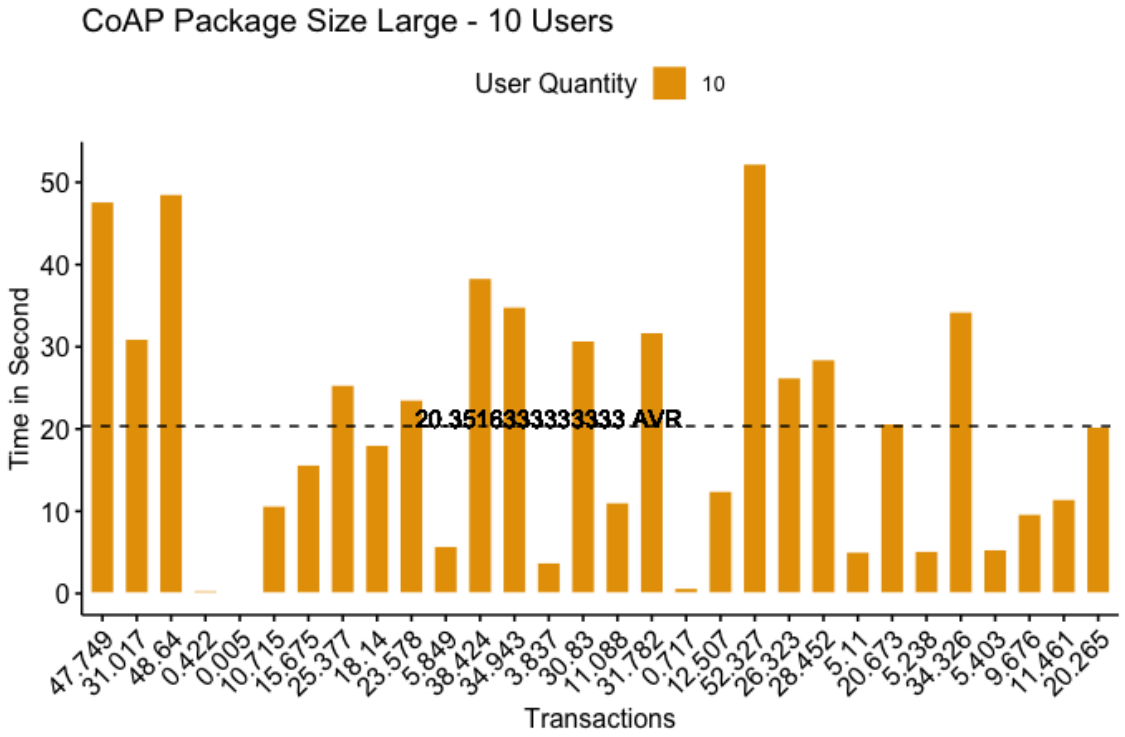


Figure A.11: CoAP Large - Transactions 100 users

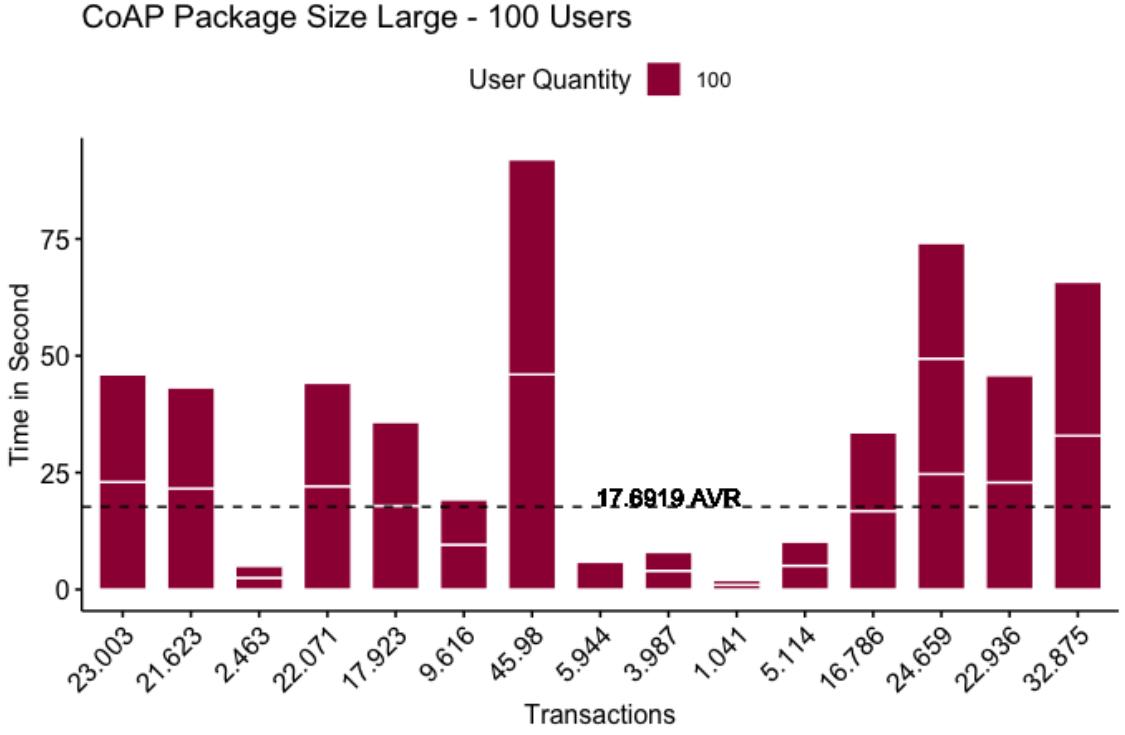
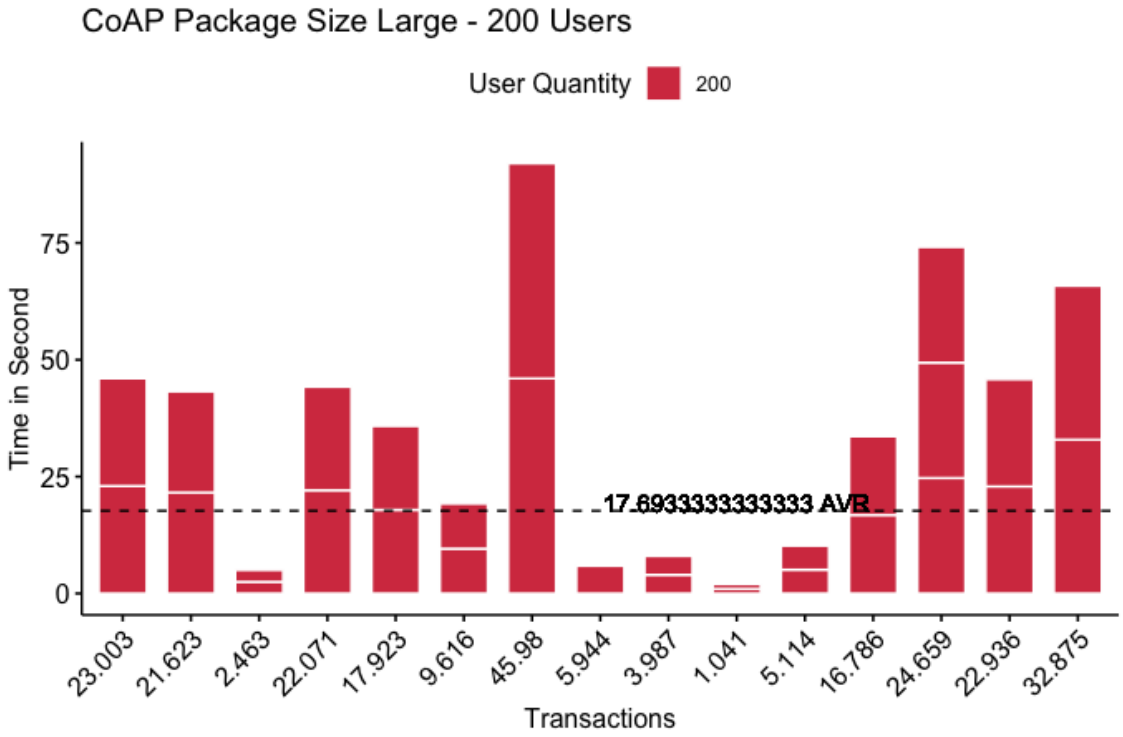


Figure A.12: CoAP Large - Transactions 200 users



A.1.4 Group I - MQTT QoS 0 Small

This scenario evaluates the total time in seconds for MQTT QoS 0 transactions with large-sized packages (20B), without latency and where DME does not give information about the protocol. Each payload size was executed 30 times following the test plan outlined in Table 5.2.

The goal of the first experiment is to obtain the total response time a metric for MQTT QoS 0 transactions with small-sized messages (20B) shown in Table A.4 and Graph A.13. The experiment is divided into numbers of users (10, 100, 200) with the configuration described in Table 5.2. The results are based on the number of users: 10 users in Graph A.14, 100 users in Graph A.15 , 200 users in Graph A.16.

Table A.4: MQTT QoS 0 Small - Total Time in Seconds

MQTT QoS 0 No Latency No DME	**Average Total Time in Seconds**
	Small (20B)
10 users	20.34
100 users	23.83
20 users	29.93

Table A.4 consolidates the total time in second displayed by three layers Edge,

Fog and Cloud. In the following graphs, the data shows the transactions by time per second, each bar represents a transactions and the transactions with the same time. When the time is equal, the transactions are displayed above the other, as shown in Graphs A.15 and A.16.

Figure A.13: MQTT QoS 0 - Total time includes all the users

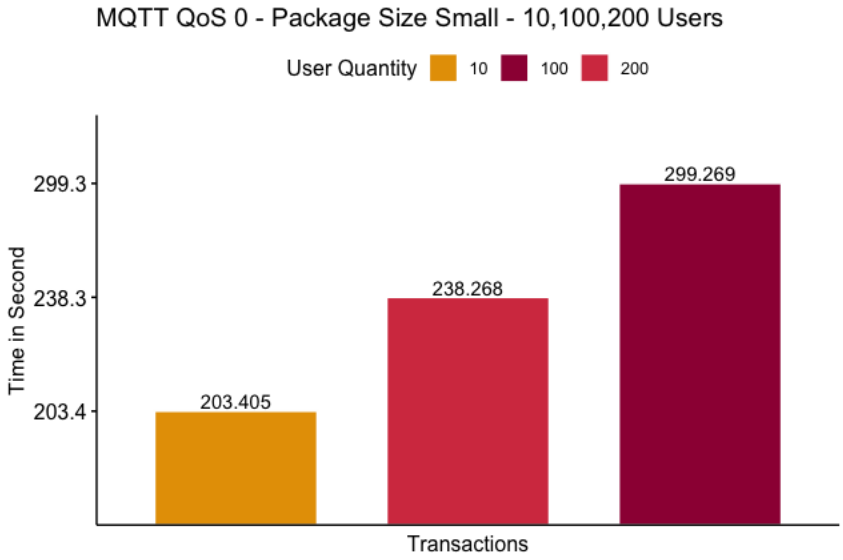


Figure A.14: MQTT QoS 0 - Transactions 10 users

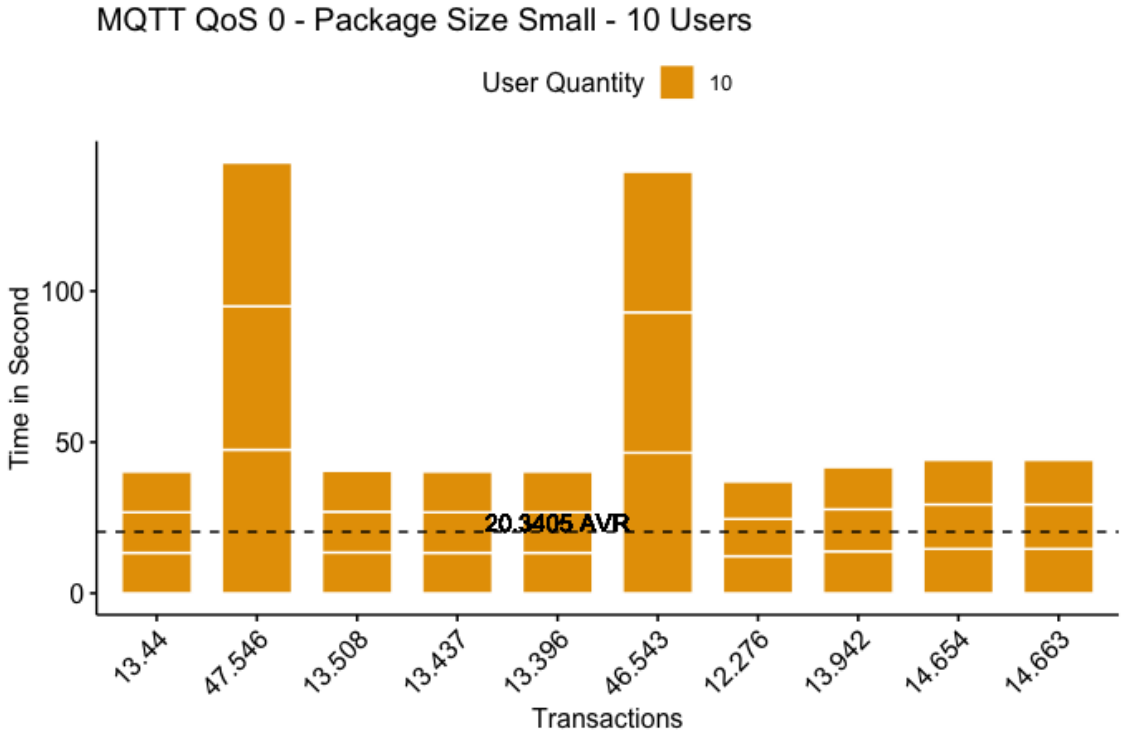


Figure A.15: MQTT QoS 0 - Transactions 100 users

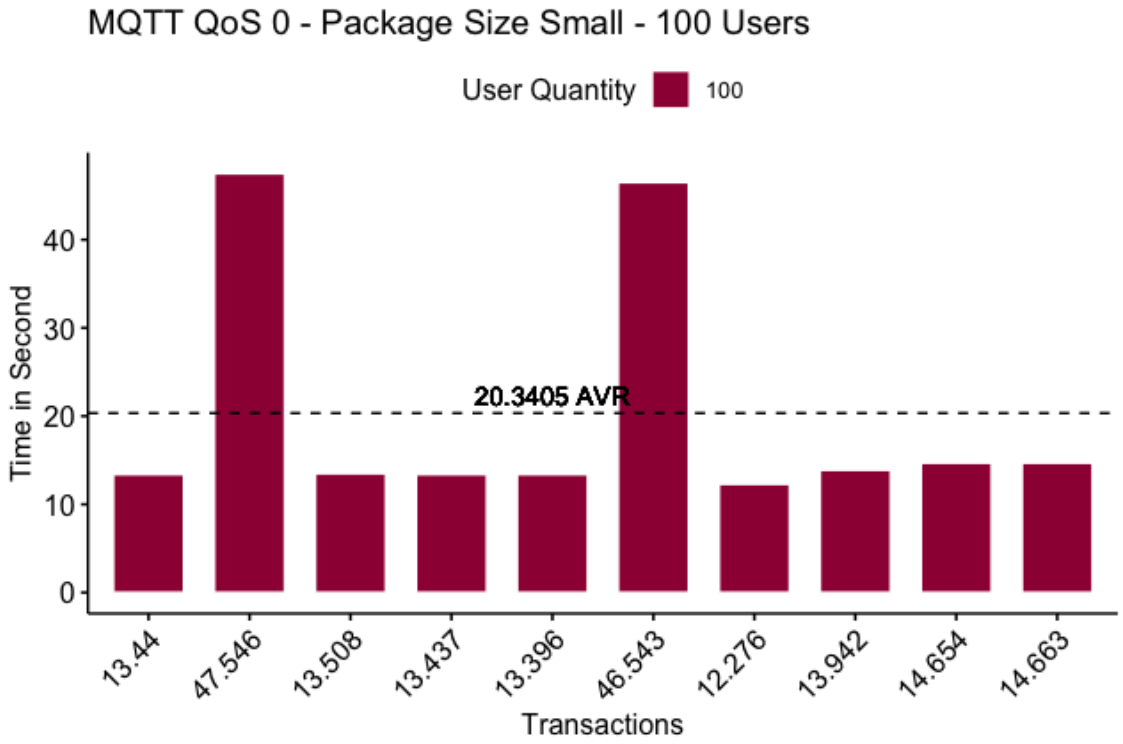
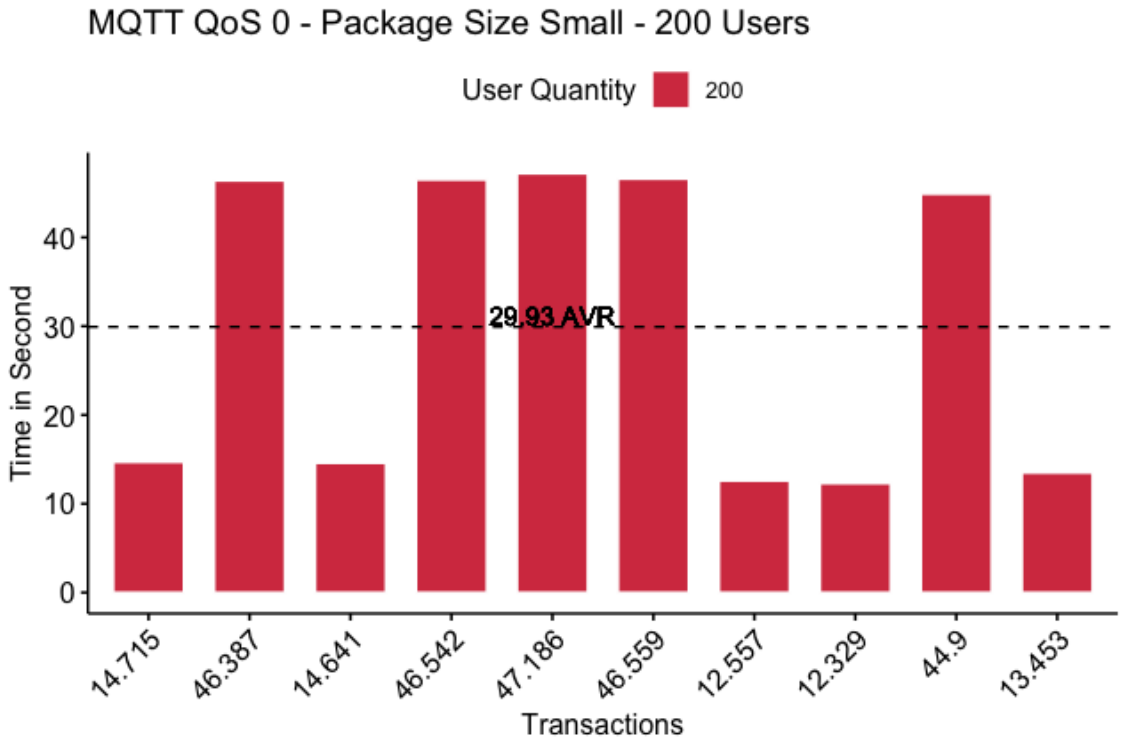


Figure A.16: MQTT QoS 0 - Transactions 200 users



A.1.5 Group I - MQTT QoS 0 Medium

This scenario evaluates the total amount of time in seconds for MQTT QoS 0 transactions with medium-sized packages (800B), without latency andin a situation where DME does not inform the protocol. Each payload size was executed 30 times following the test plan outlined in Table 5.2.

The goal of the first experiment is to obtain the total response time as a metric for MQTT QoS 0 transactions with medium-sized messages (800B) shown in Table A.5 and Graph A.17. The experiment is divided into numbers of users (10, 100, 200) with the configuration described in Table 5.2. The results are based on the number of users: 10 users are shown in Graph A.18, 100 users in Graph A.19, and 200 users in Graph A.20.

Table A.5: MQTT QoS 0 Medium - Total Time in Seconds

MQTT QoS 0 No Latency No DME	**Average Total Time in Seconds**
	Medium (800B)
10 users	20.34
100 users	23.83
20 users	29.93

Table A.5 consolidates the total time in seconds displayed by three layers Edge, Fog and Cloud. In the following graphs, the data shows the transactions by time per second each bar represents a transaction and the transactions with the same time. When the time is equal, the transactions are displayed above the other as shown in Graphs A.19 and A.16.

Figure A.17: MQTT QoS 0 Medium - Total time includes users

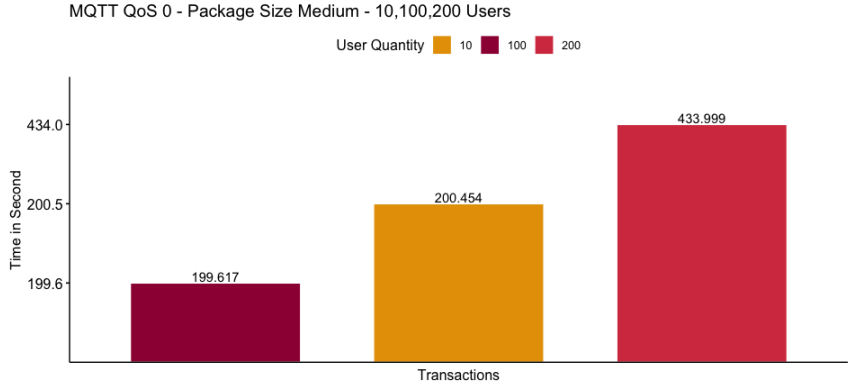


Figure A.18: MQTT QoS 0 Medium - Transactions 10 users

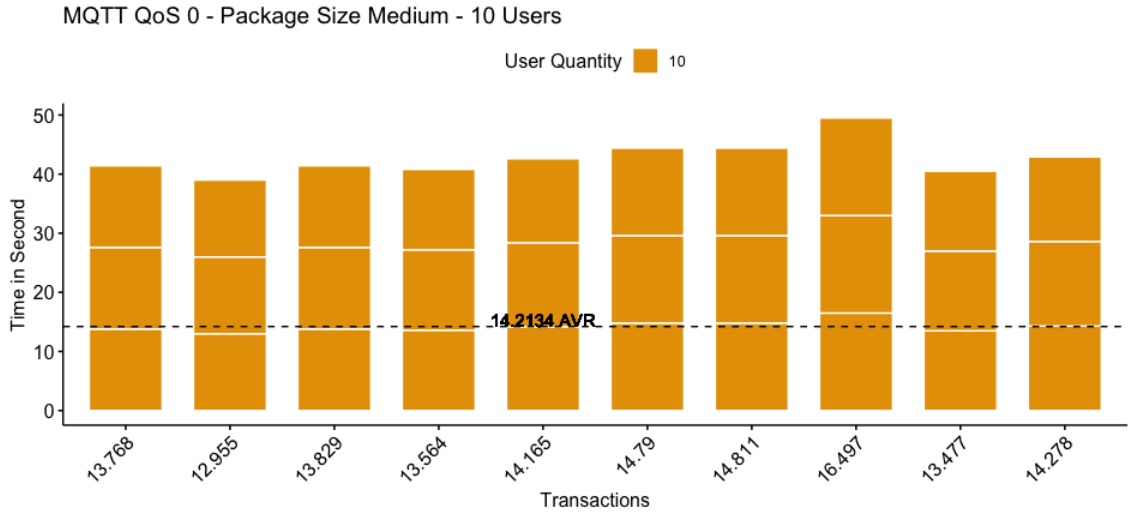
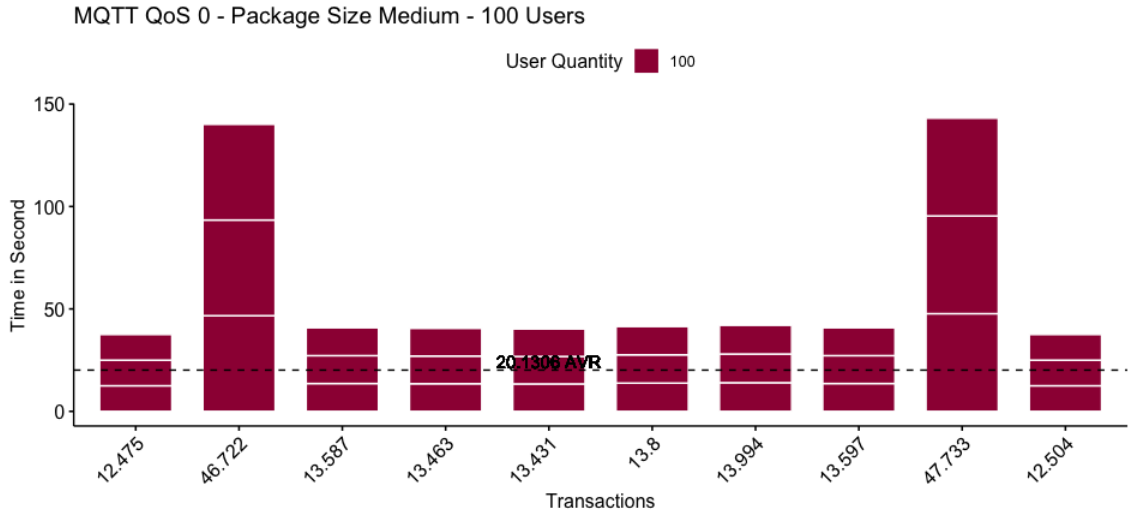


Figure A.19: MQTT QoS 0 Medium - Transactions 100 users



A.1.6 Group I - MQTT QoS 0 Large

This scenario evaluates the total time in seconds for MQTT QoS 0 transactions with a large-sized package(2048B), without latency and in a situation where DME does not inform the protocol. Each payload size was executed 30 times following the test plan outlined on Table 5.2.

The purpose of the first experiment is to obtain the total response time a metric for MQTT QoS 0 transactions with large messages size (2048B) as shown in on Table A.6 and Graph A.21. The experiment is divided into number of users (10, 100, 200) with the configuration described in Table 5.2. The results are based on the number of user: 10 users as shown in Graph A.22, 100 users in Graph A.23, and 200 users in Graph A.24.

Figure A.20: MQTT QoS 0 Medium - Transactions 200 users

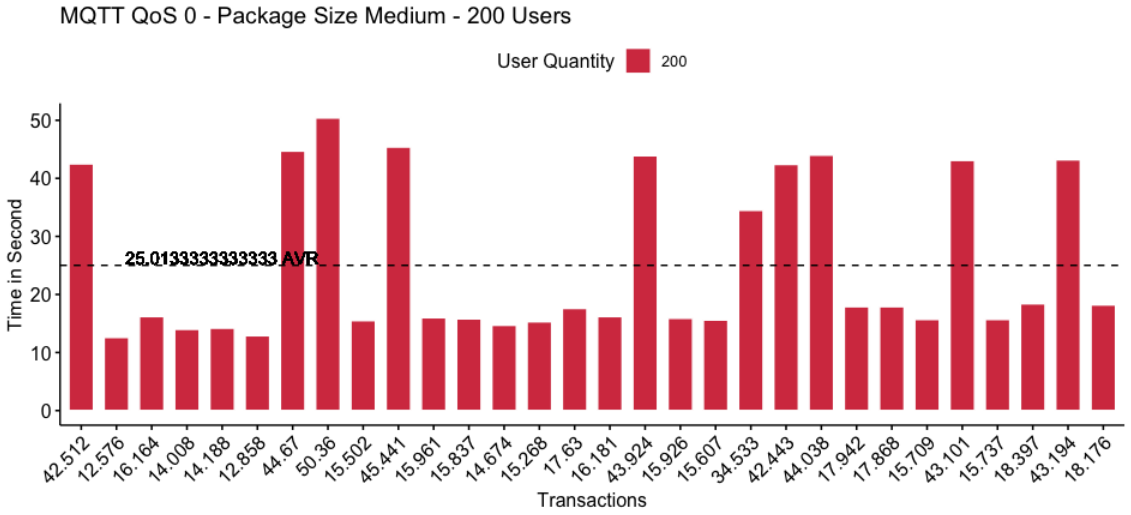


Table A.6: MQTT QoS 0 Large - Total Time in Seconds

MQTT QoS 0 No Latency No DME	**Average Total Time in Seconds**
	Large (2048B)
10 users	21.092
100 users	20.14
20 users	25.01

Table A.6 consolidates the total amount of time in seconds three layers Edge, Fog and Cloud. In the following graphs, the data shows the transactions by time per second, each bar represents a transactions and the transactions with the same time. When the time is equal, the transactions are displayed above the other as shown in Graphs A.23.

Figure A.21: MQTT QoS 0 Large - Total time all users

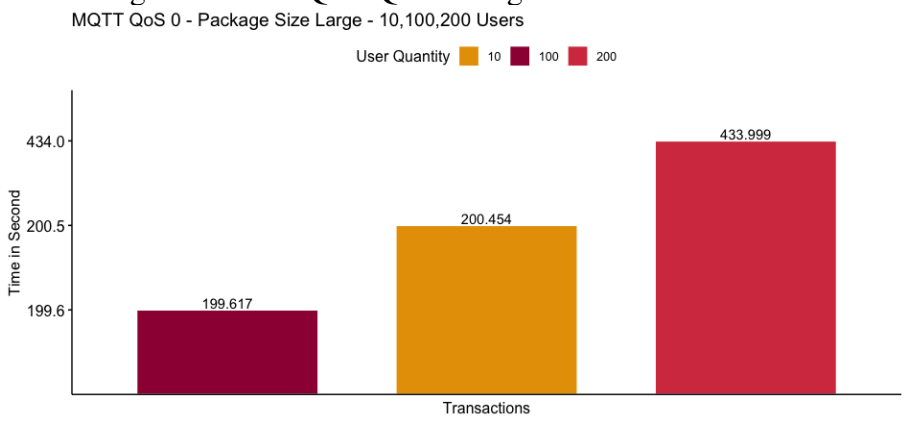


Figure A.22: MQTT QoS 0 Large - Transactions 10 users

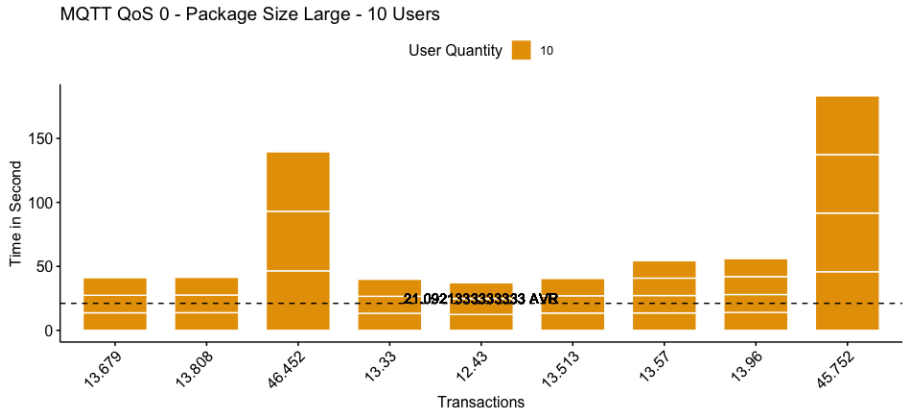


Figure A.23: MQTT QoS 0 Large - Transactions 100 users

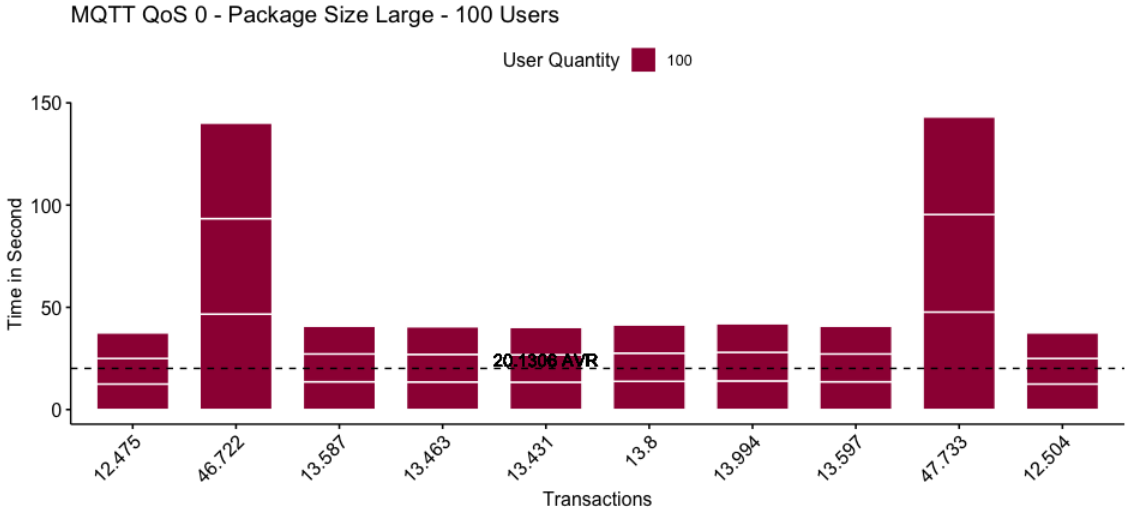
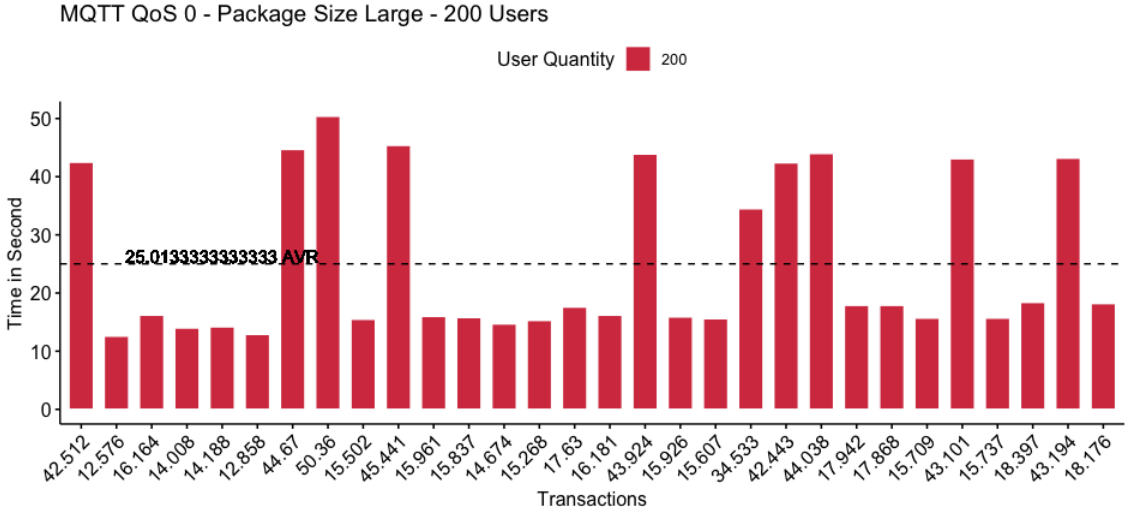


Figure A.24: MQTT QoS 0 Large - Transactions 200 users



A.1.7 Group I - MQTT QoS 1 Small

This scenario evaluates the total time in seconds for MQTT QoS 1 transactions with a small-sized package (20B), without latency and in a situation where DME does not give information the protocol. Each payload size was executed 30 times following the test plan outlined in Table 5.2.

The purpose of the first experiment is to obtain the total response time as a metric for MQTT QoS 1 transactions with small-sized messages (20B) as shown in Table A.7 and Graph A.13. The experiment is divided into number of users (10, 100, 200) with the configuration described in Table 5.2. The results are based on the number of users: 10 users in Graph A.26, 100 users in Graph A.27, 200 users in Graph A.28.

Table A.7: MQTT QoS 1 Small - Total Time in Seconds

MQTT QoS 1 No Latency No DME	**Average Total Time in Seconds**
	Small (20B)
10 users	28.357
100 users	27.572
20 users	27.7

Table A.7 consolidates the total time in seconds displayed by the group. When the time is equal, the transactions are displayed in Figure A.27.

Figure A.25: MQTT QoS 1 Small - Total time includes the all users

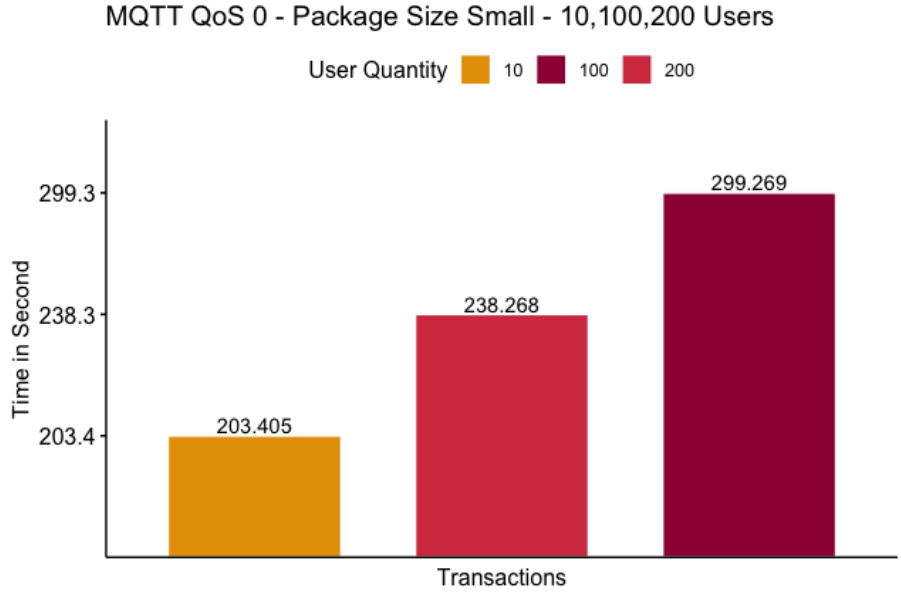


Figure A.26: MQTT QoS 1 Small - Transactions 10 users

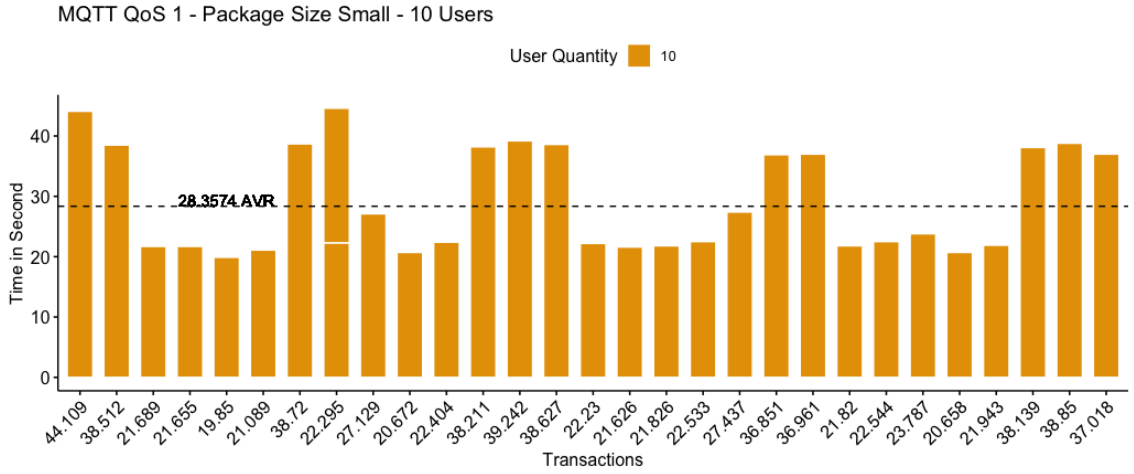


Figure A.27: MQTT QoS 1 Small - Transactions 100 users

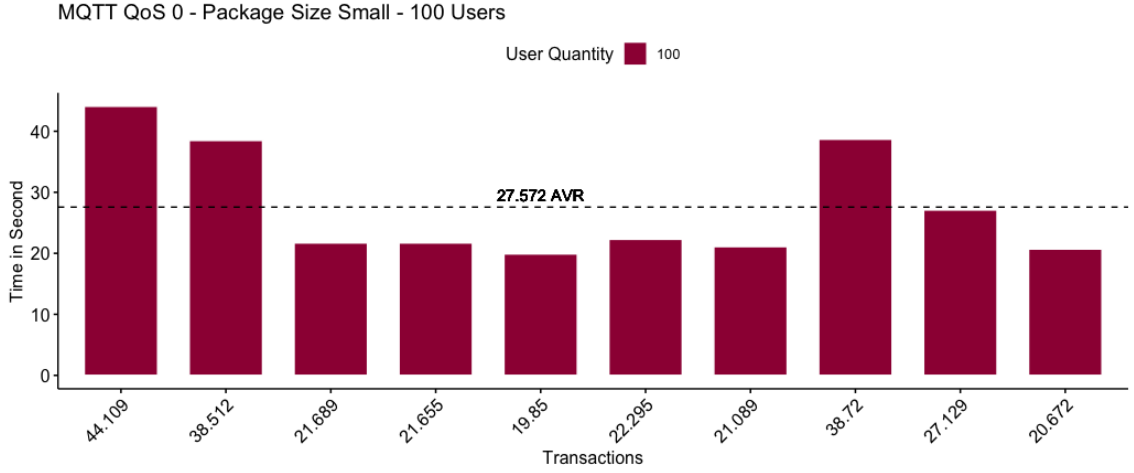
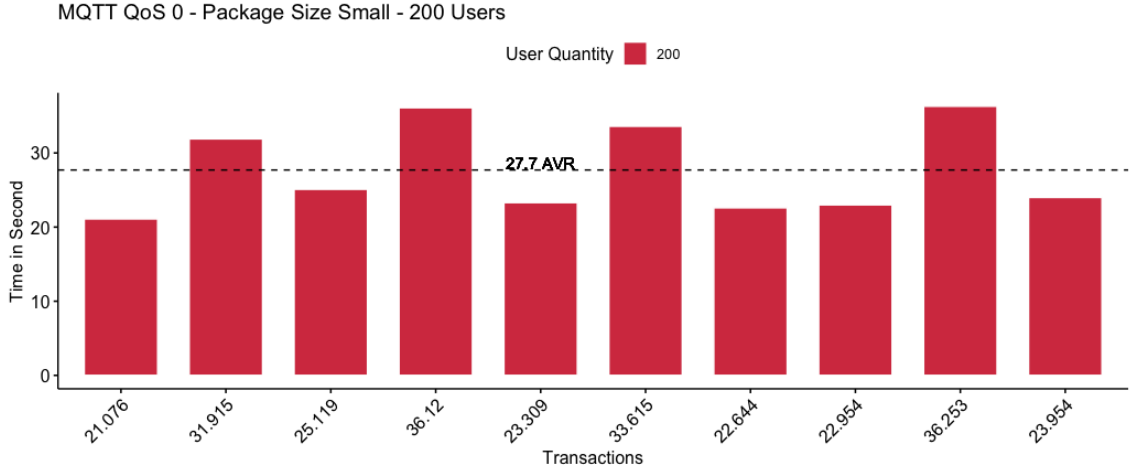


Figure A.28: MQTT QoS 1 Small - Transactions 200 users



A.1.8 Group I - MQTT QoS 1 Medium

This scenario evaluates the total time in seconds for MQTT QoS 1 transactions with a package size of 800B without applying latency. The DME does not notify the protocol. According to the test plan displayed in the Table 5.2, each payload size was executed on a total of 30 occasions.

The purpose of the first experiment is to obtain the total response time as a metric for MQTT QoS 1 transactions with medium-sized messages (800B) shown in on Table A.8 and Graph A.29. The experiment is divided into number of users (10, 100, 200) with the configuration described in Table 5.2. The results are based on the numbers of user: 10 users in Graph A.30, 100 users in Graph A.31, 200 users in Graph A.32.

Table A.8: MQTT QoS 1 Medium - Total Time in Seconds

MQTT QoS 1 No Latency No DME	**Average Total Time in Seconds**
	Medium (800B)
10 users	28.873
100 users	27.30
20 users	26.50

Table A.8 consolidates the total time in second displayed by three layers Edge, Fog and Cloud. In the following graph the data shows the transactions by time per second; each bar represents a transactions and the transactions with the same time. When the time is equal, the transactions are displayed above the other as presented on graphs A.31.

Figure A.29: MQTT QoS 1 Medium - Total time all users

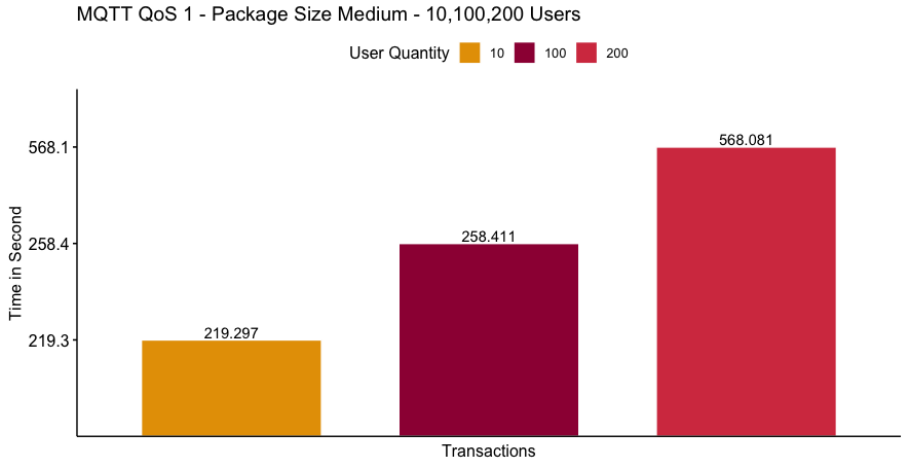


Figure A.30: MQTT QoS 1 Medium - Transactions 10 users

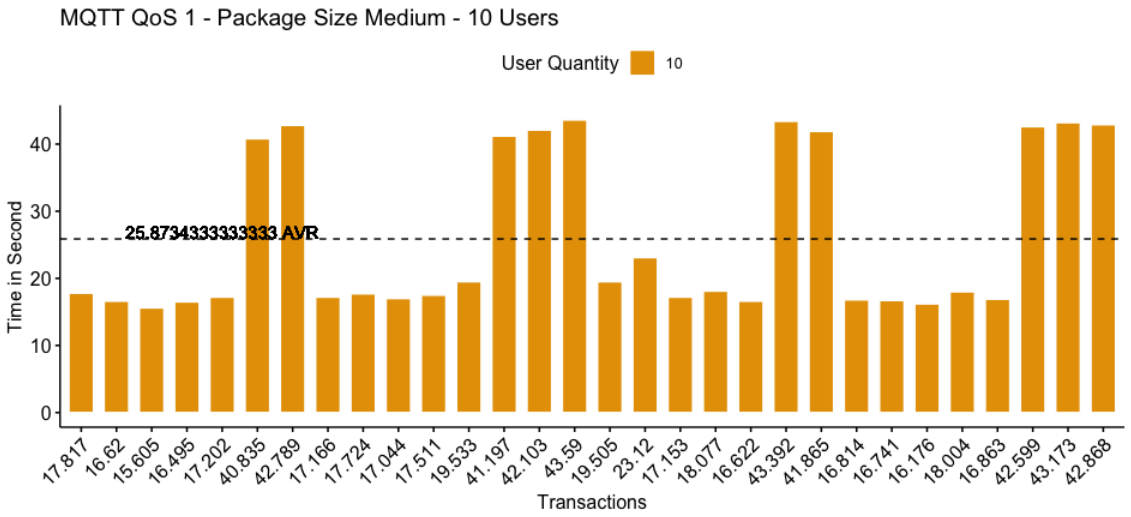


Figure A.31: MQTT QoS 1 Medium - Transactions 100 users

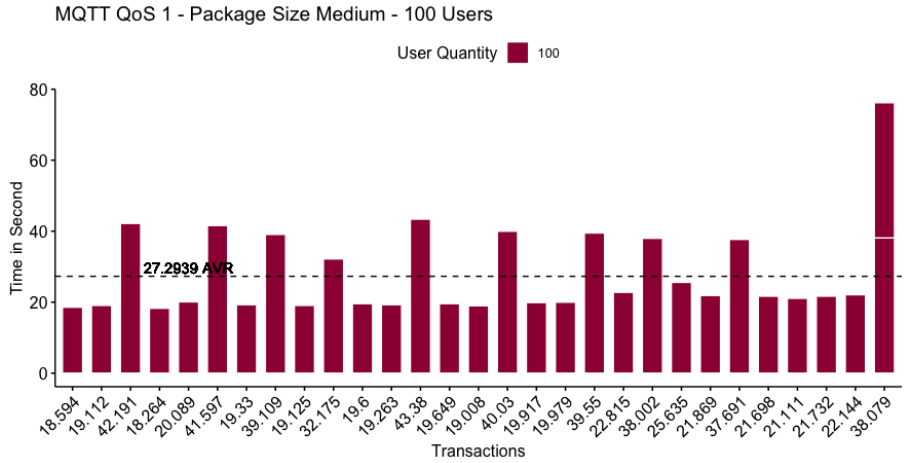
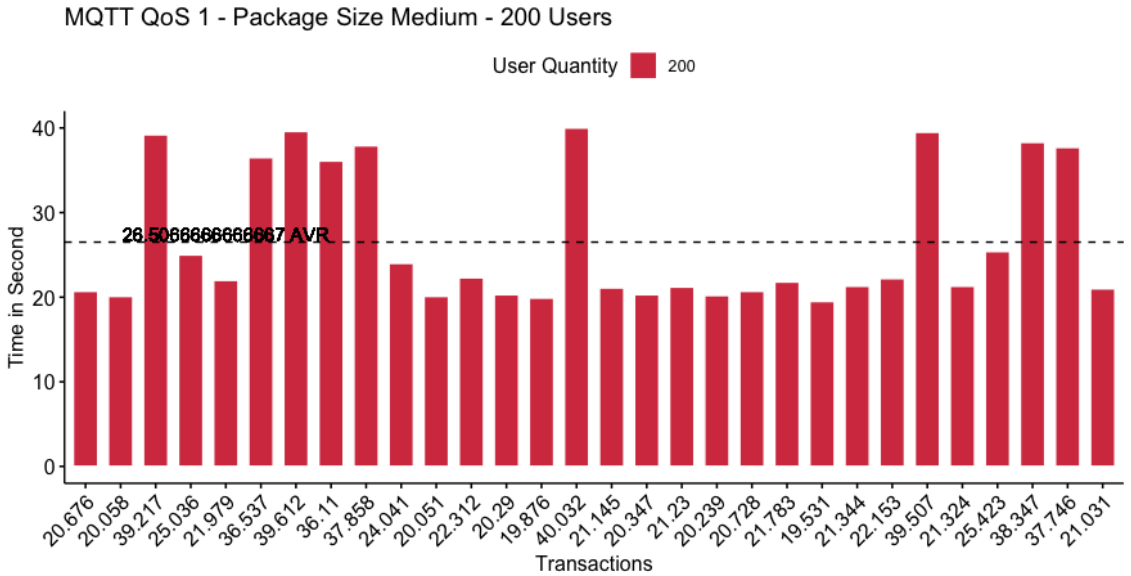


Figure A.32: MQTT QoS 1 Medium - Transactions 200 users



A.1.9 Group I - MQTT QoS 1 Large

This scenario evaluates the total time in seconds for MQTT QoS 1 transactions with a large-sized package size large (2048B), (2048B), without and in a situation where DME does not give information about the protocol. Each payload size was executed 30 times following the test plan outlined in Table 5.2.

The goal of the first experiment is to obtain the total response time as a metric for MQTT QoS 1 transactions with medium-sized messages (2048B) shown in Table A.9 below and Graph A.33. The experiment is divided into the number of users (10, 100, 200) with the configuration described in Table 5.2. The results are based on the number of users: 10 users in Graph A.34, 100 users in Graph A.35, 200 users in graph A.36.

Table A.9: MQTT QoS 1 Large - Time in Seconds

MQTT QoS 1 No Latency No DME	**Average Total Time in Seconds**
	Large (2048B)
10 users	22.89
100 users	21.67
20 users	25.01

Table A.9 consolidates the total time in seconds three layer Edge, Fog and Cloud. In the following graphs, the data shows the transactions by time per second; each bar represents a transactions and the transaction with the same time. When the time is equal, the transactions are displayed one block above the other as shown on graph A.35.

Figure A.33: MQTT QoS 1 Large - Total time for all the users

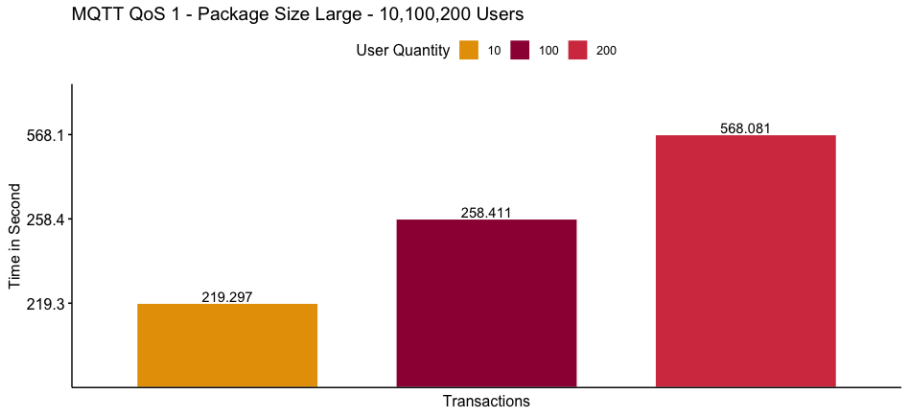


Figure A.34: MQTT QoS 1 Large - Transactions 10 users

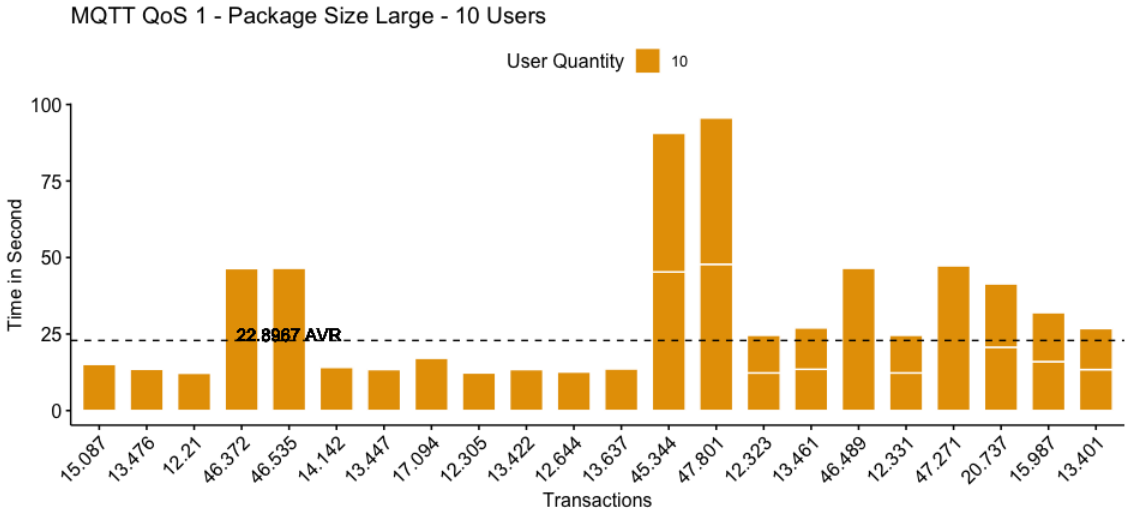


Figure A.35: MQTT QoS 1 Large - Transactions 100 users

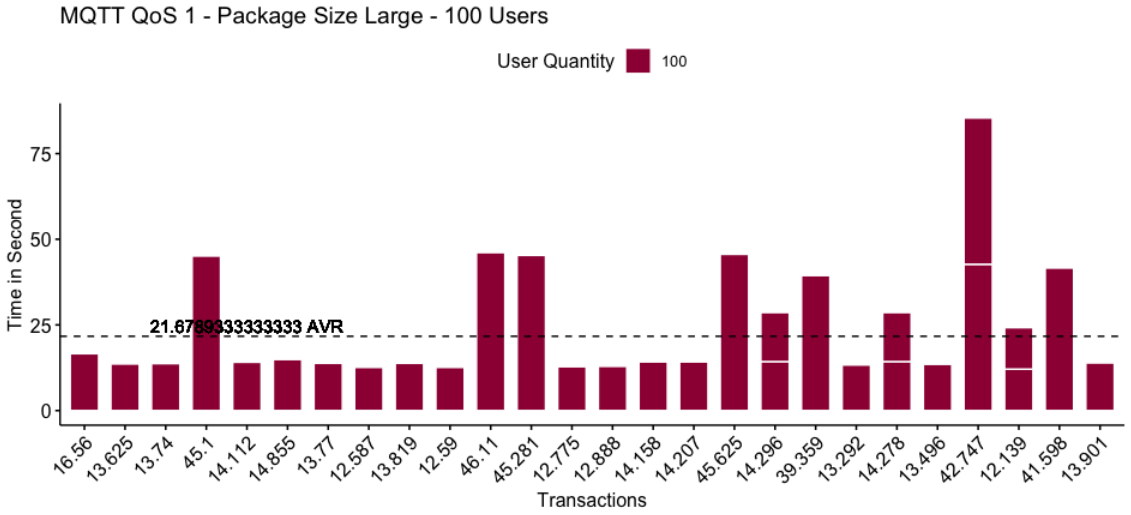
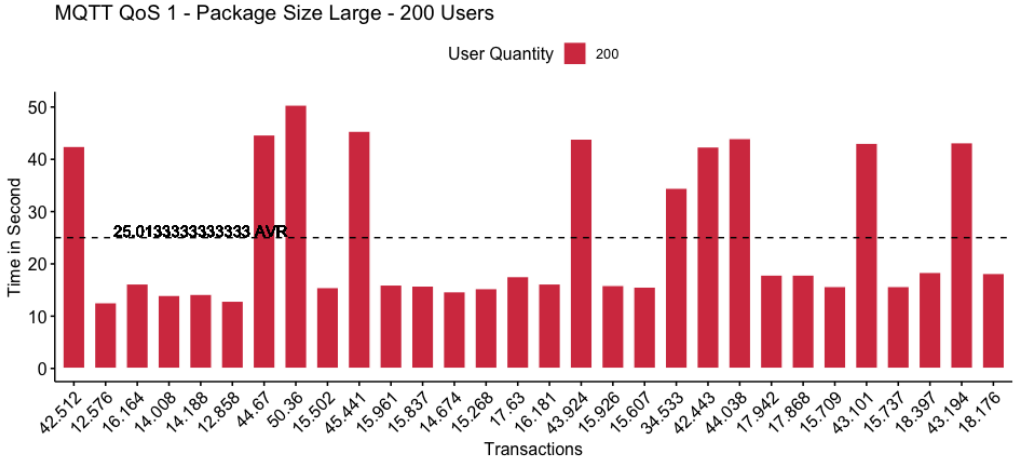


Figure A.36: MQTT QoS 1 Large - Transactions 200 users



A.1.10 Group I - MQTT QoS 2 Small

This scenario evaluates the total time in seconds for MQTT QoS TWO transactions with a small-sized package (20B), without latency and in a situation where DME does not notify the protocol. Each payload size was executed 30 times following the test plan outlined in Table 5.2.

The purpose of the first experiment is to obtain the total response time as a metric for MQTT QoS 1 transactions with small-sized messages(20B) as shown on Table A.10 and Graph A.37. The experiment is divided into the number of users (10, 100, 200) with the configuration described in Table 5.2. The results are based on the numbers of users: 10 users in Graph A.38, 100 users in Graph A.39, and 200 users in Graph A.40.

Table A.10: MQTT QoS 2 Small - Total Time in Seconds

MQTT QoS TWO No Latency No DME	**Average Total Time in Seconds**
	Small (20B)
10 users	19.63
100 users	19.18
20 users	19.79

The table A.10 consolidates the total time in second three layers Edge, Fog and Cloud. In the following graphs, the data shows the transactions by time per second; each bar represents a transactions and the transactions with the same time. When the time is equal, the transactions are displayed one block above the other as shown in Graphs A.39.

Figure A.37: MQTT QoS 2 Small - Total time all users

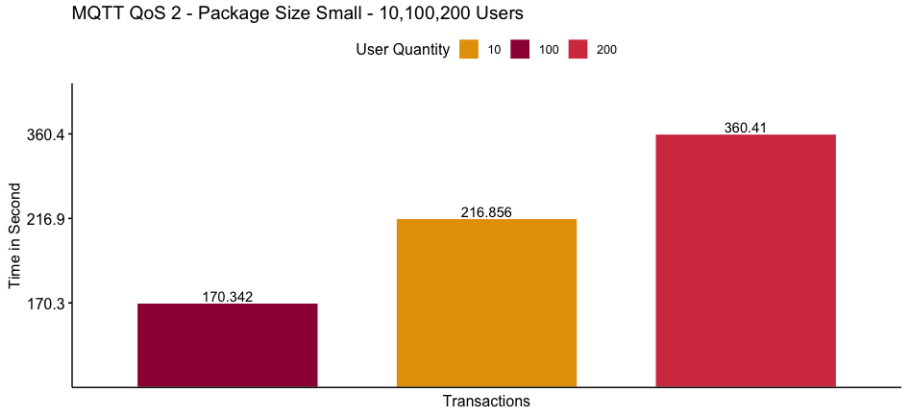


Figure A.38: MQTT QoS 2 Small - Transactions 10 users

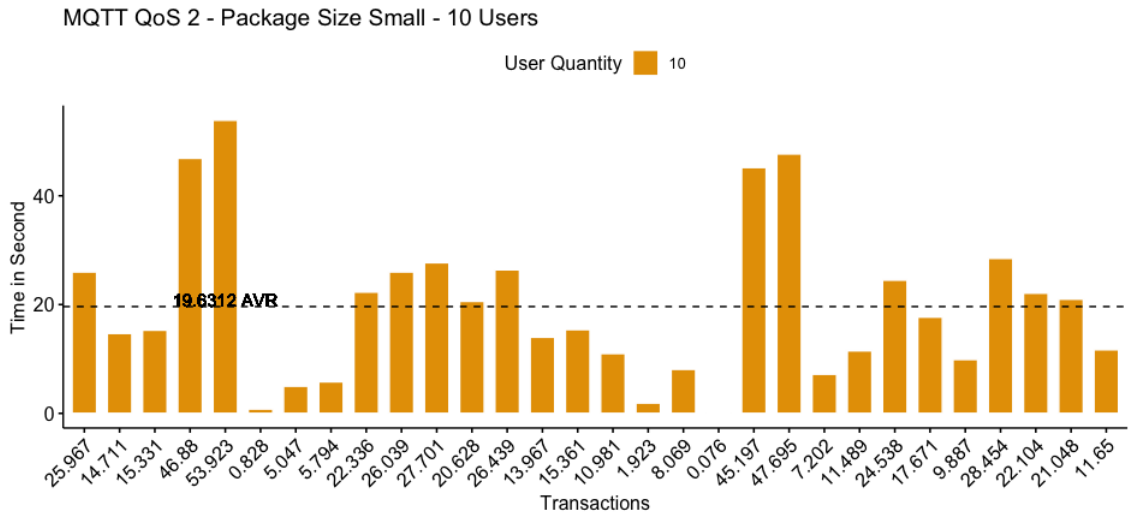


Figure A.39: MQTT QoS 2 Small - Transactions 100 users

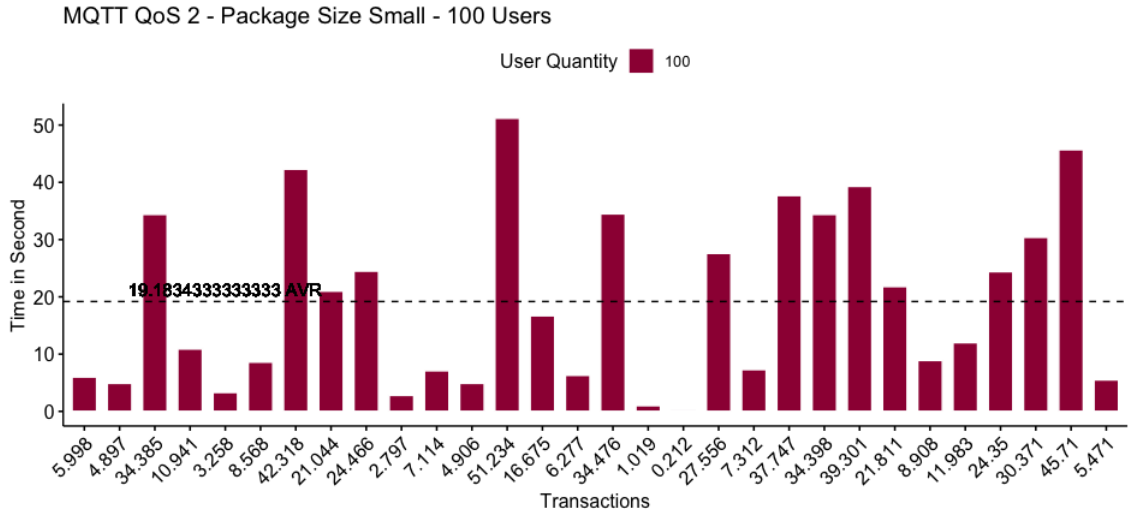
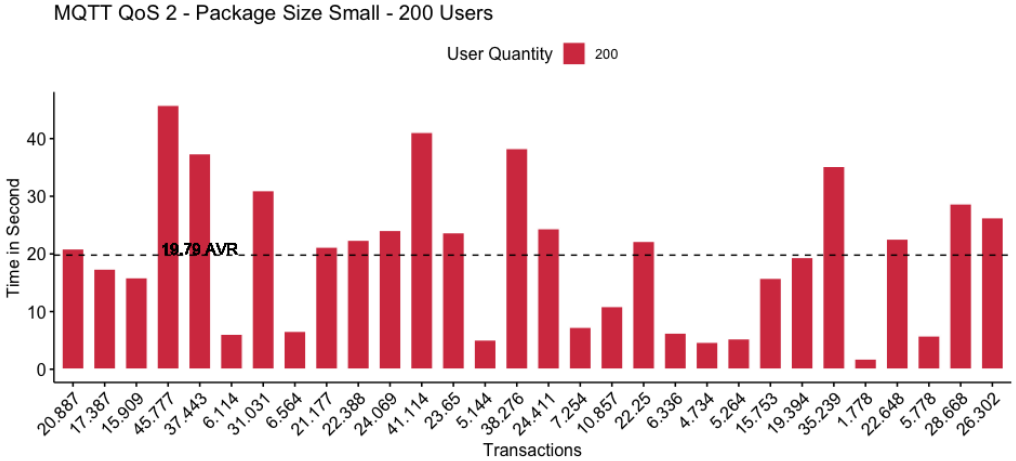


Figure A.40: MQTT QoS 2 Small - Transactions 200 users



A.1.11 Group I - MQTT QoS 2 Medium

This scenario evaluates the total time in seconds for MQTT QoS TWO transactions with a medium-sized package (800B), without latency and in a situation where DME does not notify the protocol. Each payload size was executed 30 times following the test plan outlined in Table 5.2.

The purpose of the first experiment is to obtain the total response time as a metric for MQTT QoS 2 transactions with medium-sized messages (800B) a shown in Table A.11 and Graph A.41. The experiment is divided into the number of users (10, 100, 200) with the configuration described in Table 5.2. The results are based on the number of users: 10 users in Graph A.42, 100 users is presented on graphic A.43, and 200 users in Graph A.44.

Table A.11: MQTT QoS 2 Medium - Total Time in Seconds

MQTT QoS TWO No Latency No DME	**Average Total Time in Seconds**
	Medium (800B)
10 users	20.25
100 users	19.52
20 users	25.13

Table A.11 consolidates the total time in seconds three layers Edge, Fog and Cloud. In the following graphs, the data shows the transactions by time per second; each bar represent a transactions and the transactions with the same time. When the time is equal, the transactions are displayed one block above the other as shown in Graph A.43.

Figure A.41: MQTT QoS 2 Medium - Total time for all the users

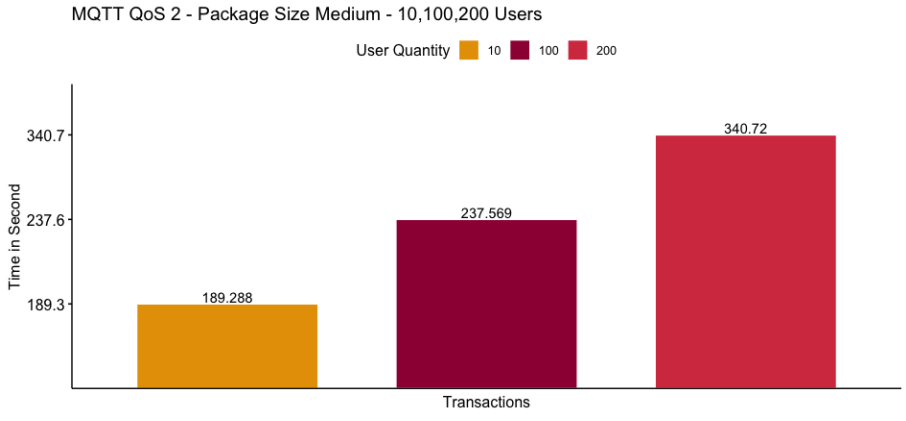


Figure A.42: MQTT QoS 2 Medium - Transactions 10 users

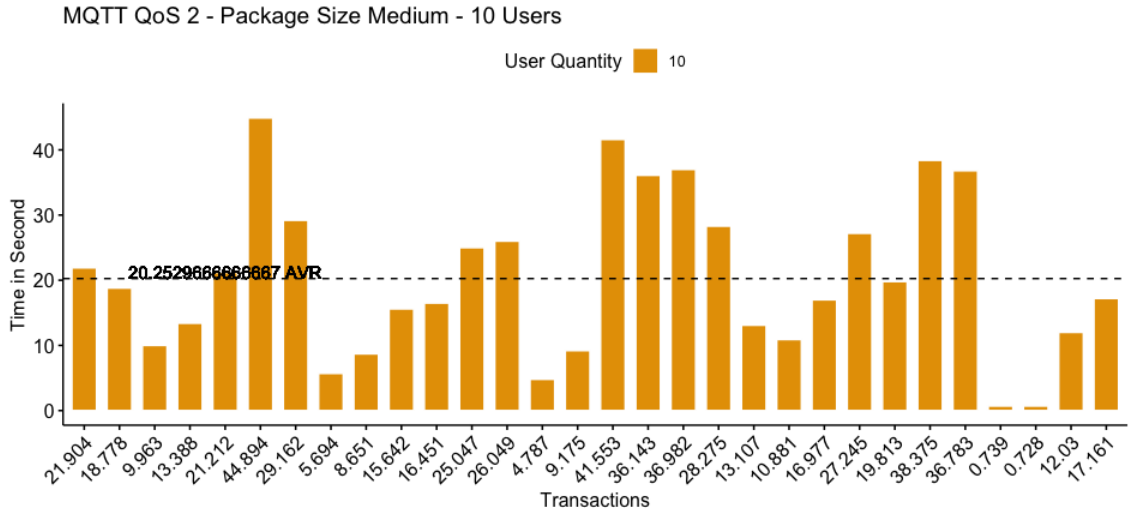


Figure A.43: MQTT QoS 2 Medium - Transactions 100 users

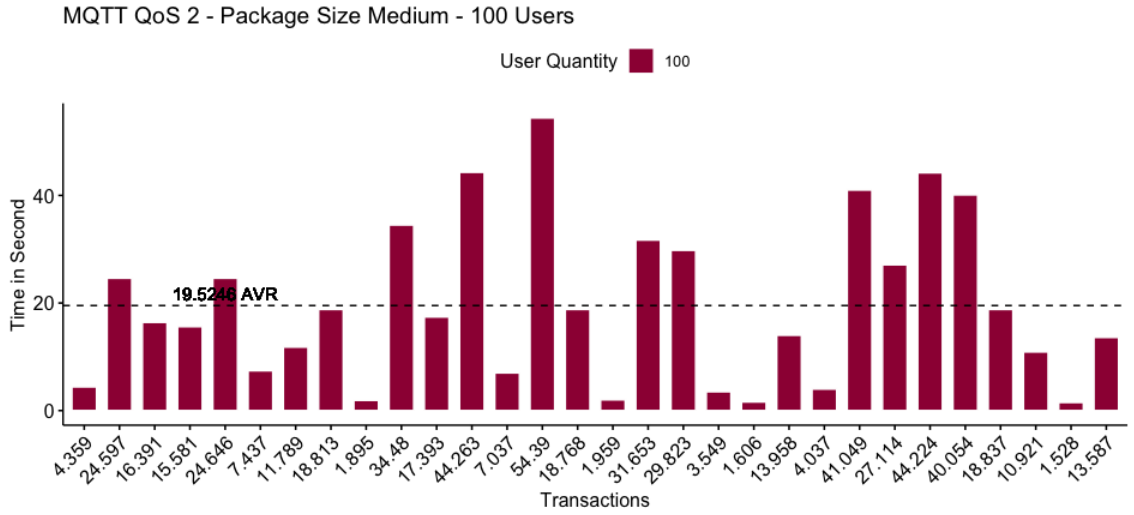
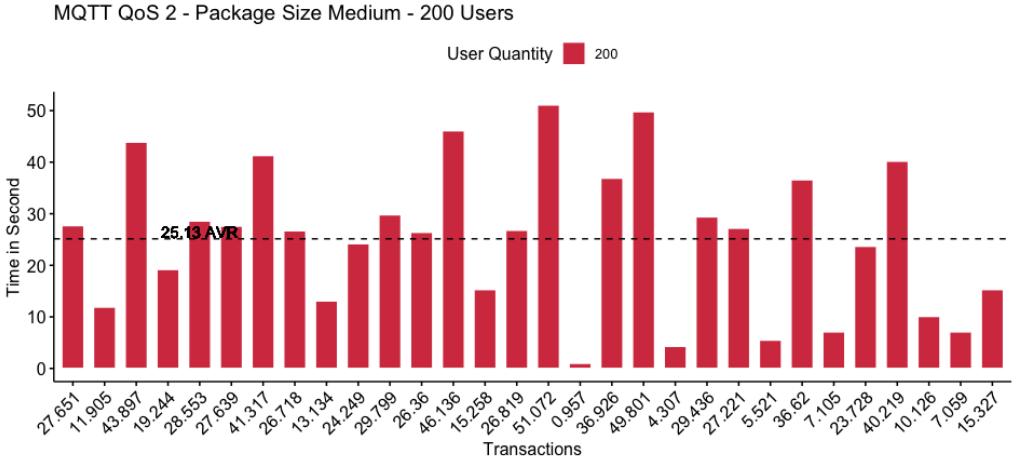


Figure A.44: MQTT QoS 2 Medium - Transactions 200 users



A.1.12 Group I - MQTT QoS 2 Large

This scenario evaluates the total time in seconds for MQTT QoS TWO transactions with package size large (2048B), without apply latency and DME does not inform the protocol. Each payload size was executed 30 times following the test plan detailed on table 5.2.

The purpose of the first experiment is to obtain the total response time as a metric for MQTT QoS TWO transactions with medium-sized messages (2048B), as shown in Table A.12 and Graph A.45. The experiment is divided into the number of users (10, 100, 200) with the configuration described in Table A.2. The results are based on the number of users: 10 users in Graph A.46, 100 users in Graph A.47, 200 users in Graph A.48.

Table A.12: MQTT QoS 2 Medium - Total Time in Seconds

MQTT QoS TWO No Latency No DME	**Average Total Time in Seconds**
	Large (2048B)
10 users	19.63
100 users	19.18
20 users	19.79

Table A.12 consolidates the total time in seconds three layers Edge, Fog and Cloud. In the following graphs, the data shows the transactions by time per second; each bar represent a transaction and the transactions with the same time. When the time is equal, the transactions are displayed one block above the other as shown in Graphs A.47.

Figure A.45: MQTT QoS 2 Large - Total time all users

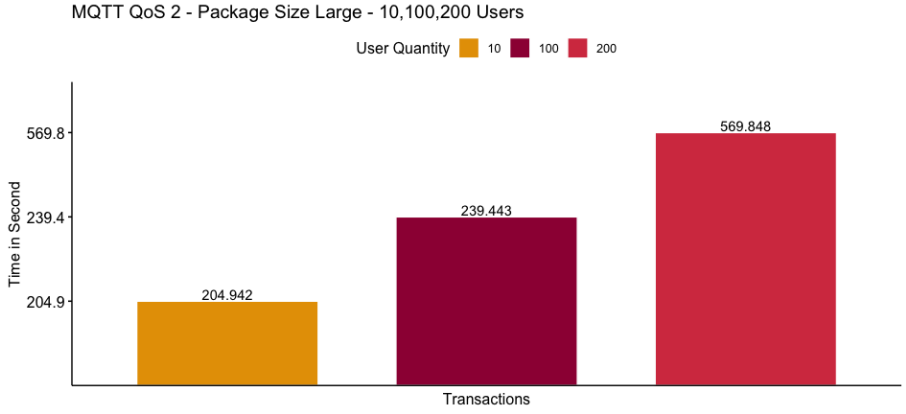


Figure A.46: MQTT QoS 2 Large - Transactions 10 users

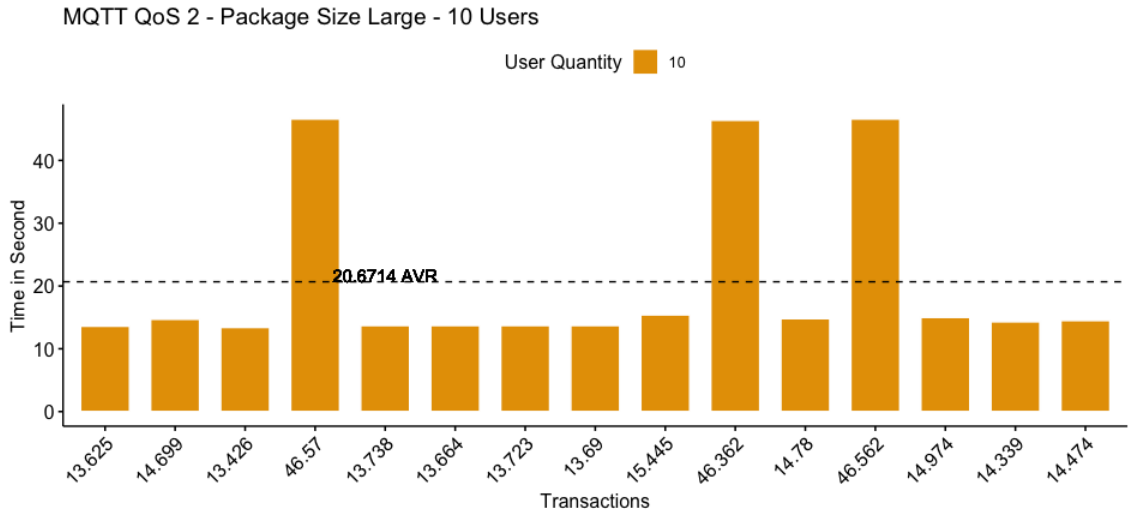


Figure A.47: MQTT QoS 2 Large - Transactions 100 users

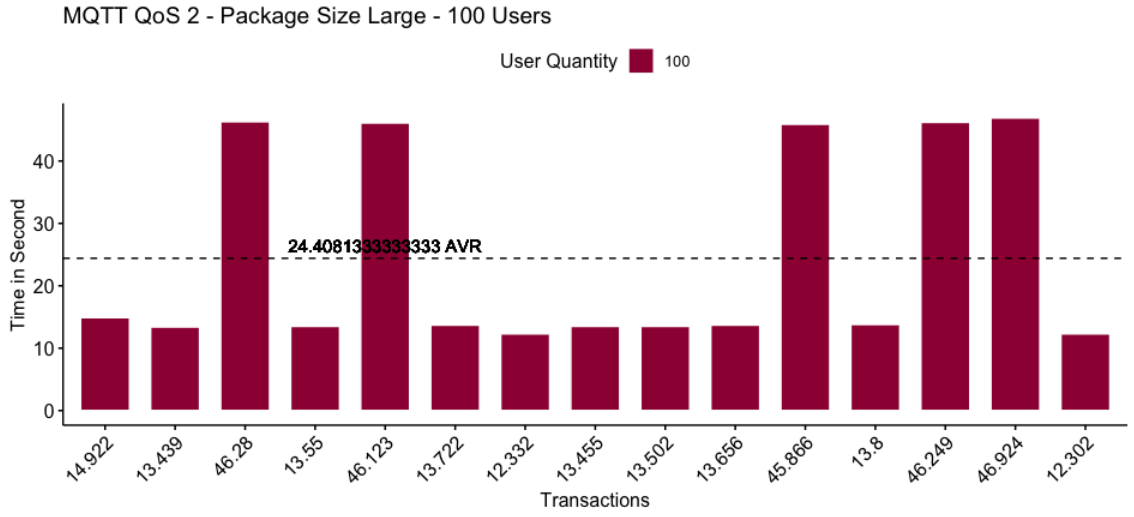
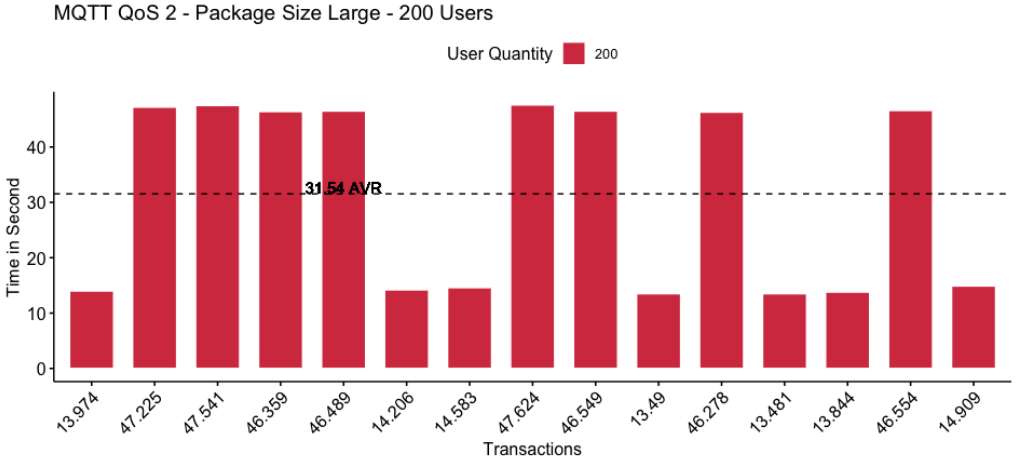


Figure A.48: MQTT QoS 2 Large - Transactions 200 users



APPENDIX B — RESUMO EXPANDIDO

A Internet das Coisas (IoT) tem um papel fundamental na transformação digital da sociedade, indo além do uso de dispositivos inteligentes. Ela representa a transição do mundo analógico para o digital, permitindo que decisões mais eficientes sejam tomadas com base em dados obtidos em tempo real. A crescente demanda por redes interconectadas de sensores e dispositivos tem exigido infraestruturas robustas e eficientes, capazes de processar, analisar e fornecer respostas rápidas e precisas aos usuários. Dentro deste contexto, um dos desafios mais críticos que emergem é a latência de comunicação entre dispositivos IoT, que pode impactar diretamente a performance e a confiabilidade das soluções baseadas em IoT.

Com o aumento da quantidade de dados gerados por dispositivos IoT, surge a necessidade de implementar arquiteturas que possam otimizar o tráfego de informações e reduzir a latência. O conceito de MiddleFog surge como uma solução intermediária entre os dispositivos IoT e a nuvem, focando na mitigação da latência através do gerenciamento eficiente das mensagens de comunicação. Este trabalho se propõe a investigar esse desafio, explorando como o MiddleFog pode melhorar a transmissão de dados e otimizar a seleção de protocolos de comunicação em tempo real.

A latência de comunicação é um dos maiores desafios enfrentados na implementação de soluções IoT em larga escala. Em uma rede de IoT, o tempo que uma mensagem leva para viajar entre um dispositivo e seu destino pode ser determinante para o sucesso ou falha da aplicação. O atraso na comunicação pode ocorrer por diversos motivos, incluindo congestionamento da rede, baixa eficiência no processamento de pacotes e a distância física entre os dispositivos IoT e os servidores na nuvem.

A latência se torna especialmente problemática em aplicações críticas, como em sistemas de saúde, veículos autônomos ou cidades inteligentes, onde atrasos mínimos podem causar consequências significativas. Nesse contexto, é necessário encontrar soluções que possam minimizar esses impactos e garantir que as mensagens sejam transmitidas de forma eficiente e confiável.

O MiddleFog é uma camada intermediária posicionada entre os dispositivos IoT e a nuvem, atuando como um middleware que melhora o gerenciamento da comunicação. A principal funcionalidade do MiddleFog é permitir a seleção dinâmica do protocolo de comunicação mais apropriado, com base nas condições da rede. Entre os protocolos utilizados estão o MQTT (Message Queuing Telemetry Transport) e o CoAP (Constrained

Application Protocol), que são amplamente utilizados em ambientes IoT devido à sua leveza e eficiência.

A arquitetura do MiddleFog é composta por uma estrutura que monitora constantemente o estado da rede e, com base nisso, faz a seleção do protocolo mais adequado para garantir a melhor performance possível. Por exemplo, em situações onde a latência é baixa e a rede estável, o MQTT pode ser escolhido para garantir uma comunicação eficiente entre os dispositivos. Em contrapartida, em ambientes onde a latência é alta e há perdas frequentes de pacotes, o CoAP pode ser mais apropriado, por sua capacidade de funcionar de maneira otimizada em redes com restrições.

Além de otimizar a seleção de protocolos, o MiddleFog também atua na redução das limitações de comunicação entre o Multi-access Edge Computing (MEC) e a nuvem, minimizando os impactos de latência, perda de pacotes e baixa eficiência da rede. Ao descentralizar parte do processamento para a camada Fog, o MiddleFog consegue responder de forma mais rápida às solicitações dos dispositivos, sem a necessidade de enviar todas as informações para a nuvem, o que reduz significativamente o tempo de resposta.

As avaliações realizadas com o MiddleFog mostraram resultados promissores na mitigação de problemas de latência e perda de pacotes. Em testes realizados, foi observada uma taxa de perda de mensagens de até 25% quando a comunicação se dá diretamente entre dispositivos IoT e a nuvem, sem a utilização da camada Fog. No entanto, ao incorporar o MiddleFog como intermediário na comunicação, essa taxa foi significativamente reduzida, resultando em uma melhoria no desempenho de até 48

A utilização do MiddleFog demonstra que a inclusão de uma camada intermediária entre dispositivos IoT e a nuvem pode não apenas melhorar a confiabilidade da comunicação, mas também aumentar a eficiência geral do sistema. Ao permitir que o tráfego de dados seja gerenciado de maneira mais inteligente, com uma seleção dinâmica de protocolos e descentralização do processamento, o MiddleFog se apresenta como uma solução viável para otimizar as infraestruturas IoT em ambientes com alta demanda de dados.

O MiddleFog é uma solução escalável e modular que contribui para a mitigação de um dos desafios na implementação de soluções IoT em larga escala: a latência de comunicação. Ao posicionar-se como uma camada intermediária entre dispositivos IoT e a nuvem, o MiddleFog otimiza a transmissão de dados e melhora a seleção de protocolos de comunicação, adaptando-se às condições da rede em tempo real. Isso não só garante uma melhor performance e confiabilidade do sistema, como também minimiza os impactos causados pela latência, perda de pacotes e ineficiência da rede.

As avaliações demonstraram que a utilização do MiddleFog pode reduzir significativamente a taxa de perda de mensagens e aumentar a eficiência da comunicação entre dispositivos IoT. Esses resultados mostram que soluções baseadas em middleware, como o MiddleFog, podem ser fundamentais para o futuro da IoT, onde a demanda por redes rápidas, confiáveis e eficientes continuará a crescer.