Al-Quds Open University

Faculty of Graduate Studies and Scientific Research

Master of Information Technology

Comparative Analysis of IoT Protocols Efficiency for Smart City Scenarios: Performance Challenges

تحليل مقارن لكفاءة بروتوكولات إنترنت الأشياء لسيناريوهات المدينة الذكية: تحديات الأداء

THESIS

by

Ahmad Awni Khalaf

Student ID: 0330012110183

Supervisor

Dr. Eng. Samer Hosni Jaloudi

Submitted in Partial Fulfillment of the Requirements

For the Degree of Master of Information Technology at the Faculty of Graduate Studies

Ramallah, Palestine

Al-Quds Open University

Faculty of Graduate Studies and Scientific Research

Master of Information Technology

Comparative Analysis of IoT Protocols Efficiency for Smart City Scenarios: Performance Challenges

تحليل مقارن لكفاءة بروتوكولات إنترنت الأشياء لسيناريوهات المدينة الذكية: تحديات الأداء

THESIS

by

Ahmad Awni Khalaf

Student ID: 0330012110183

Supervisor

Dr. Eng. Samer Hosni Jaloudi

Submitted in Partial Fulfillment of the Requirements

For the Degree of Master of Information Technology at the Faculty of Graduate Studies

Ramallah, Palestine

Examination Committee Page

The committee for

Ahmad Awni Ahmad Khalaf

certifies that this is the approved version of the following thesis and is acceptable in quality and form for publication in paper and digital formats:

Comparative Analysis of IoT Protocols Efficiency for Smart City Scenarios: Performance Challenges

Committee Members:

Committee Supervisor: [insert name]

Signature: ____

Date: ____

Committee Co-Supervisor (if appropriate): [insert name]

Signature: ____

Date: ____

Committee First Member: [insert name]

Signature: ____

Date: ____

Committee Second Member: [insert name]

Signature: ____

Date: ____

Committee Third Member (if appropriate): [insert name]

Signature: ____

Date: ____

Al-Quds Open University

2025

Declaration

I, Ahmad Awni Khalaf, hereby declare that the work presented in this thesis has not been submitted for any other degree or professional qualification, and that it is the result of my independent work.

Signed:

Date:

Abstract

Comparative Analysis of IoT Protocols Efficiency for Smart City Scenarios:

Performance Challenges

The Internet of Things has transformed modern life by changing how humans, systems, and devices interact with the physical world. The number of IoT devices is expected to increase from 10 billion in 2020 to approximately 30 billion in 2030. With the increasing reliance on the Internet of Things (IoT) as a primary technology for developing smart cities, several technical challenges have emerged due to the heterogeneous and distributed nature of IoT networks. One of the most critical challenges lies in choosing an efficient communication protocol. It is a complex mission and needs more effort from the developers. Therefore, one of the primary objectives of this thesis is to examine the performance of application layer protocols for the Internet of Things in the context of smart city development. The research work in this thesis is organized into two parts: a theoretical part and a practical part. A detailed literature review was applied as a methodology in the theoretical phase of this study. The strengths and weaknesses of five existing IoT application layer protocols—HTTP, MQTT, CoAP, XMPP, and AMQP—in smart city IoT environments were evaluated, and a comparative performance analysis was conducted. The philosophy of each protocol has been addressed separately regarding its applicability in the smart city scenario. Based on the results obtained from the theoretical phase, MQTT emerges as an effective and ideal compromise protocol that balances the strengths and limitations of these protocols in terms of performance and efficiency; it combines lightweight and ease of use with an acceptable level of reliability and quality of service. Since the MQTT protocol has been suggested as the main communication protocol in IoT, the performance of this protocol was evaluated under various scenarios in the context of smart city applications, focusing on two primary scenarios: the transmission of small payloads and large payloads, each evaluated under varying QoS levels (0, 1, and 2). Using a Raspberry Pi as the publisher and Hive MQ as the broker, key performance metrics, including latency and message loss, were analyzed to assess protocol efficiency. Ultimately, the study emphasizes the importance of selecting an appropriate protocol tailored to the specific requirements of smart city applications, underscoring that no single solution is suitable for all IoT deployments. Additionally, the conclusions drawn from this extensive review revealed that various factors can impact a protocol evaluation.

الخلاصة

تحليل مقارن لكفاءة بروتوكولات إنترنت الأشياء لسيناريوهات المدينة الذكية: تحديات الأداء

لقد أحدثت إنترنت الأشياء نقلة نوعية في الحياة العصرية من خلال تغيير كيفية تفاعل البشر والأنظمة والأجهزة مع العالم المادي. ومن المتوقع أن يزداد عدد أجهزة إنترنت الأشياء من 10 مليارات في عام 2020 إلى ما يقرب من 30 مليارًا في عام 2030. ومع الاعتماد المتزايد على إنترنت الأشياء كتكنولوجيا أساسية لتطوير المدن الذكية، ظهرت العديد من التحديات التقنية بسبب الطبيعة غير المتجانسة والموزعة لشبكات إنترنت الأشياء. يكمن أحد أهم التحديات في اختيار بروتوكول اتصال فعال. إنها مهمة معقدة وتتطلب المزيد من الجهد من المطورين. لذلك، فإن أحد الأهداف الرئيسية لهذه الرسالة هو التحقيق في أداء بروتوكولات طبقة التطبيق لإنترنت الأشياء في تطوير المدينة الذكية. يتم تنظيم العمل البحثي في هذه الرسالة إلى جزأين: جزء نظري وجزء عملي. تم تطبيق مراجعة الأدبيات كمنهجية في المرحلة النظرية من هذه الدراسة. تم تقييم نقاط القوة والضعف لخمسة بروتوكولات موجودة لطبقة تطبيقات إنترنت الأشياء - HTTP و MQTT و CoAP و XMPP و - AMQPفي بيئات إنترنت الأشياء للمدن الذكية، وأُجري تحليل أداء مقارن. تمت معالجة فلسفة كل بروتوكول على حدة فيما يتعلق بإمكانية تطبيقه في سيناريو المدينة الذكية. بناءً على النتائج التي تم الحصول عليها من المرحلة النظرية، يظهر MQTT كبروتوكول حل وسط فعال ومثالي يوازن بين نقاط القوة والقيود لهذه البروتوكولات من حيث الأداء والكفاءة؛ فهو يجمع بين خفة الوزن وسهولة الاستخدام مع مستوى مقبول من الموثوقية وجودة الخدمة. نظرًا لأن بروتوكول MQTT قد تم اقتراحه كبروتوكول الاتصال الرئيسي في إنترنت الأشياء، فقد تم تقييم أداء هذا البروتوكول في ظل سيناريوهات مختلفة في سياق تطبيقات المدينة الذكية، مع التركيز على سيناريوهين رئيسيين: نقل الحمولات الصغيرة والحمولات الكبيرة، حيث تم تقييم كل منهما في ظل مستويات جودة الخدمة المتفاوتة (0 و 1 و 2). باستخدام Raspberry Pi كناشر وHiveMQ كوسيط، تم تحليل مقاييس الأداء الرئيسية، بما في ذلك زمن الوصول وفقدان الرسائل، لتقييم كفاءة البروتوكول. وأخيراً، تُركز الدراسة على أهمية اختيار بروتوكول مُناسب مُصمم خصيصًا لتلبية المتطلبات الخاصة بتطبيقات المدن الذكية، مُؤكدةً أنه لا يوجد حل واحد يُناسب جميع تطبيقات إنترنت الأشياء. إضافةً إلى ذلك، كشفت الاستنتاجات المُستخلصة من هذه المراجعة المُوسعة عن وجود عوامل مُختلفة يُمكن أن تُؤثر على تقييم البروتوكول.

Acknowledgements

In the name of Allah, Most Gracious, Most Merciful

This is my graduation thesis in the Master's Program in Information Technology at Quds Open University.

First of all, I would like to express my sincere gratitude to my thesis supervisor, Dr. Eng. Samer Hosni Jaloudi, not only for graciously accepting to supervise my work but also for his continuous support, guidance, and invaluable advice throughout the completion of this thesis. He has provided me with motivation, guidance, and valuable suggestions for my thesis throughout my final year at QOU.

I would also like to express my deep gratitude to all the professors who provided me with valuable insights on various topics along my master's journey, and for sharing their knowledge. Among the professors is Prof. Eng. Yousef S. Abuzir, Dr. Eng. Waleed Al-Salous, Dr. Eng. Samer Hosni Jaloudi, Dr. Mohamed Mahmoud Dweib, Dr. Eng. Nael Mohammed Abu Halaweh, and Dr. Thabit Suleiman Sabbah. Thanks to Al-Quds Open University for providing the students with the opportunity to participate in this master’s program. Also, my thanks to my parents, my wife, my family, and my friends for their endless support.

Ahmad Khalaf

Ramallah, Al-Quds Open University

May, 2025

Dedication

To my beloved parents,

thank you for your love and support.

To my dear wife,

whose patience, love, and encouragement gave me strength at every step.

To my son and daughters,

you are my joy and my greatest motivation.

This work is dedicated to you all, with all my heart.

Table of Contents

List of Figures

List of Tables

List of abbreviations

ACK	Acknowledgement
AMQP	Advanced Message Queuing Protocol
BLE	Bluetooth Low Energy
CoAP	Constrained Applications Protocol
DDS	Data Distribution Service
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IoT	Internet of Things
JSON	JavaScript Object Notation
LTE	Long-Term Evolution
MQTT	Message Queuing Telemetry Transport
ms	Millisecond
QoS	Quality of service
REST	Representational State Transfer
RTT	Round-trip time
TCP/IP	Transmission Control Protocol/Internet Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator
WiFi	Wireless fidelity
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol
WSN	Wireless Sensor Network

Chapter 1: Introduction

This chapter aims to provide an overview and background of the thesis, including the motivation, problem statement, research objectives, the importance of the study, and the contributions made to the field. The concept of IoT, IoT architecture, and core components are presented.

1.1 Overview and Background

The Internet of Things is a network of devices that can communicate with each other through different protocols. Things include devices and gadgets, or maybe appliances such as lights, headphones, cell phones, fridges, thermostats, cameras, and even people who have implanted heart monitors. The IoT has many definitions in the field of science. The International Telecommunication Union’s (ITU) definition states that the IoT is "a global infrastructure for the information society, enabling advanced services by interconnecting things based on existing and evolving interoperable information and communication technologies." (ITU, 2022). It is a global network of physical objects with computing capabilities that collect data and transfer it to more powerful computing back-ends via communication protocols and standards. It converts everything around us into a smart object. These objects have electronics, sensors, and actuators built in, as well as software and network connectivity that let people interact with them directly.

One of the main objectives of IoT is to be 100% connected to create a smart world. That's why the transformation happens quickly; this transformation will occur in the future and will contribute to the development of several fields, such as AI integration, edge computing, cybersecurity prioritization, decentralized energy systems, a smarter urban landscape, citizen-centric solutions, and the 5G revolution. IoT is one of the popular concepts at this time in wireless communications.

The advantages of IoT can be employed in a wide range of areas, including the public and business sectors, and it has been proposed for many applications, including smart cities, consumer devices, industrial environments, the IoVT, 5G communications, and multimedia systems. 5G technology will be an important player in the development of smart cities. Wireless connectivity for smart sensing devices will be quick and safe through twinning between the Internet of Things and 5G technologies. IoT architecture defines the integrated framework of components, devices, technologies, and cloud services used to build and manage IoT systems.

It is a framework that outlines the structure and organization of the components used to build IoT solutions. It guarantees efficient communication and interoperability between the different devices and objects. IoT architecture typically consists of several layers, each with a unique objective in the overall functionality of the system. The Network IoT Architecture Perspective is focused on establishing network communication between devices or things. The main challenge is finding an IoT platform that matches the suitable application. (Guth et al., 2016). Due to their heterogeneous nature, IoT networks utilize different communication technologies, such as Wi-Fi, wireless sensor Networks, Wireless Mesh Networks, vehicular Networks, and mobile networks from 3G to 5G (Kanellopoulos et al., 2023).

The very first IoT architecture was a three-layer architecture. Figure 1.1 illustrates three-layer model architectures for IoT systems; it is composed of a perception layer as a base layer, which includes sensors and actuators, and a network layer. It is also known as the transmission layer. It acts as a bridge between the perception and application layers, which mainly realizes the transmission of information, routing, and control. The third layer is the application layer and the information processing layer, which enable user interaction. It uses cloud computing, cloud storage, and cloud analysis to integrate and monitor the data.

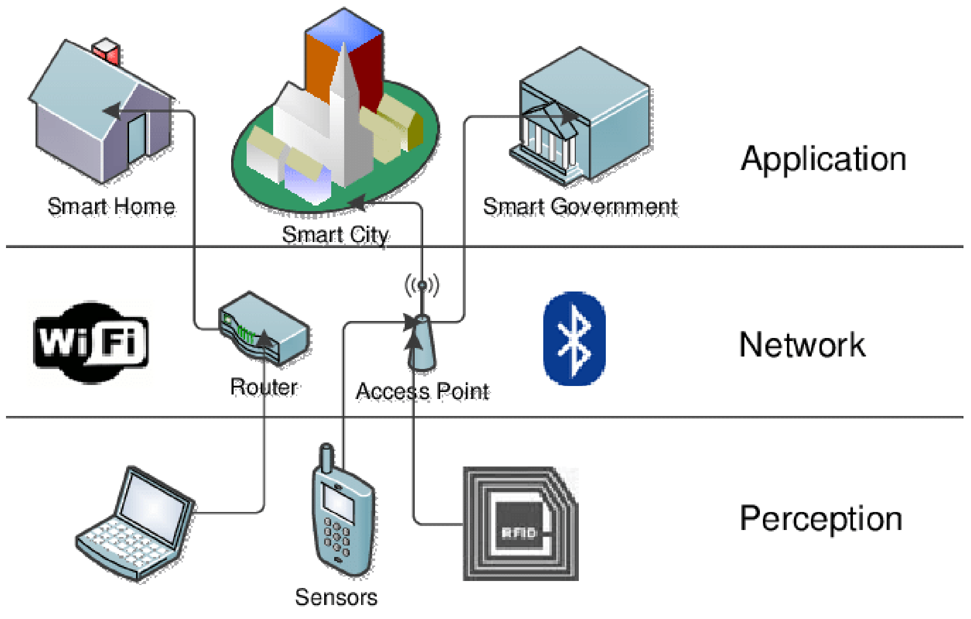


Figure 1.1: Overview of IoT 3-Layer architecture. (adapted from Yousuf, T et al., 2015)

After that, it was extended by integrating a business layer to complete the four-layer model. There are also five-layer architectures, namely the perception layer, network layer, processing layer, application layer, and business layer. The basic building blocks of an IoT system are its components, which collaborate to create systems and accomplish a variety of objectives. Each IoT component provides a function within the system in order to accomplish its objective. The data flow between IoT components may be bidirectional; some may only transmit data, while others may only receive data.

According to a research article (Raj & Shetty, 2021), the fundamental parts of an IoT ecosystem are the user interface, data processing, actuators, gateway, and sensors. (Paolone et al., 2022) proposed a definition of an IoT ecosystem as “an IoT ecosystem connects resource-constrained heterogeneous devices in a handled way to build an efficient and secure system, whose final aim is to deliver services of practical utility to a community comprising a multitude of stakeholders." In the study of (Bansal & Kumar, 2020), for example, it is stated that an IoT ecosystem combines heterogeneous components in a controlled way to create an efficient and secure system.

The IoT architecture mainly consists of the following five components, as shown in Figure 1.2 Smart devices and sensors that collect information from their surroundings and send it to the next layer which is the gateway that control the traffic between different networks, provide a certain amount of security and transfer data that comes from the sensors or is sent to the actuators, and make sure of the interoperability of the connected devices and sensors. The third one is the cloud, which provides the means and resources to gather, process, and store massive amounts of data. Analytics is the process of converting analog data from billions of smart devices and sensors into meaningful information that can be interpreted and applied to various types of analysis. The user interface is the fifth component; it serves as the visible part of the IoT system through which users can access, control, and collect information.

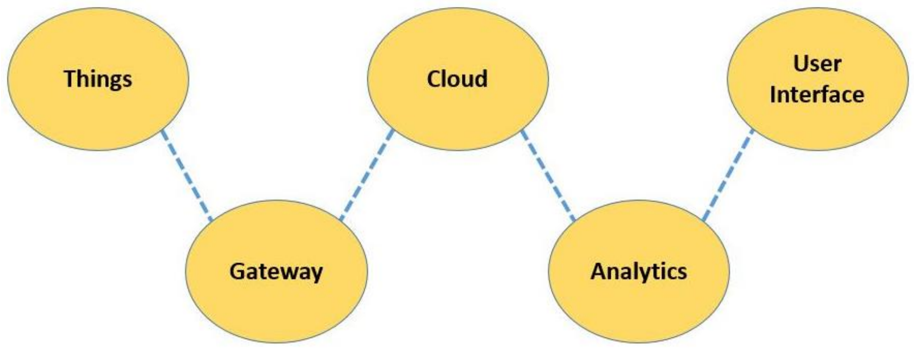


Figure 1.2: Major IoT components

The recent proposal for the IoT architecture is delivered by Cisco as a seven-layer architecture (Marcela et al., 2020). As depicted in Figure 1.3, Layer 1 represents things, physical devices, including cameras, actuators, sensors, and other devices that collect data from the physical world around us. Layer 2 focuses on communication and connectivity, for example, transferring data to cloud data centers from devices like sensors. The activities for Edge computing in layer 3 focus on converting network data into information suitable for higher-level processing and storage that is convenient for level 4 (storage and higher processing). The function of level 4 is to guarantee that the data is moving precisely. After the data accumulation is necessary to render the data and store this information in a way that facilitates the development of the application more simply and with higher performance. This is performed at layer 5. The application level is Layer 6, where all the data generated is interpreted using various types of applications. This includes the dashboards that display configuration, status, and other information about the device. Layer 7 (Collaboration and Processes), which is the highest level, pulls everything together.

The Internet of Things (IoT) and smart cities are on a path that will contribute to more connected, efficient, and sustainable urban environments in the future. The Internet of Things has a variety of applications within smart cities, such as smart buildings, smart lighting, smart water networks, smart grids, waste management using smart garbage containers, and healthcare, which can monitor health metrics such as heart rate and blood sugar levels.

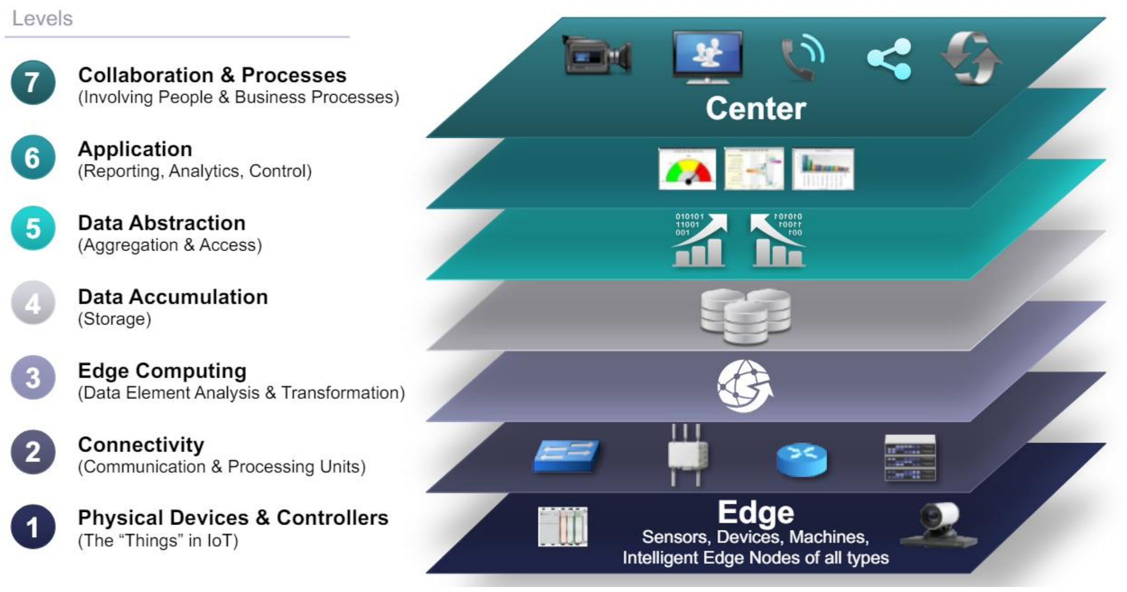

Levels
7 Collaboration & Processes
(Involving People & Business Processes)
6 Application
(Reporting, Analytics, Control)
5 Data Abstraction
(Aggregation & Access)
4 Data Accumulation
(Storage)
3 Edge Computing
(Data Element Analysis & Transformation)
2 Connectivity
(Communication & Processing Units)
1 Physical Devices & Controllers
(The "Things" in IoT)
Center
Edge
Sensors, Devices, Machines,
Intelligent Edge Nodes of all types

Figure 1.3: Overview of IoT 7-Layer architecture (Cisco, 2014)

Dameri introduces the definition of smart cities as:" A smart city is a well-defined geographical area in which high technologies such as ICT, logistics, energy production, and so on, cooperate to create benefits for citizens in terms of well-being, inclusion and participation, environmental quality, and intelligent development; it is governed by a well-defined pool of subjects, able to state the rules and policies for the city government and development" (Dameri, 2013). In the context of smart cities, the Internet of Things (IoT) offers hundreds of features, tools, and applications that significantly enhance the quality of life for residents. Figure 1.4 shows IoT components relevant to smart cities.

The number of urban residents is constantly growing, and due to the increasing urban population and the depletion of conventional resources, a rapid increase in demand for smart city initiatives.

The smart city concept serves as an effective tool for addressing contemporary urban challenges. However, there are obstacles in the way of becoming a smart city, and overcoming them will demand continuous effort.

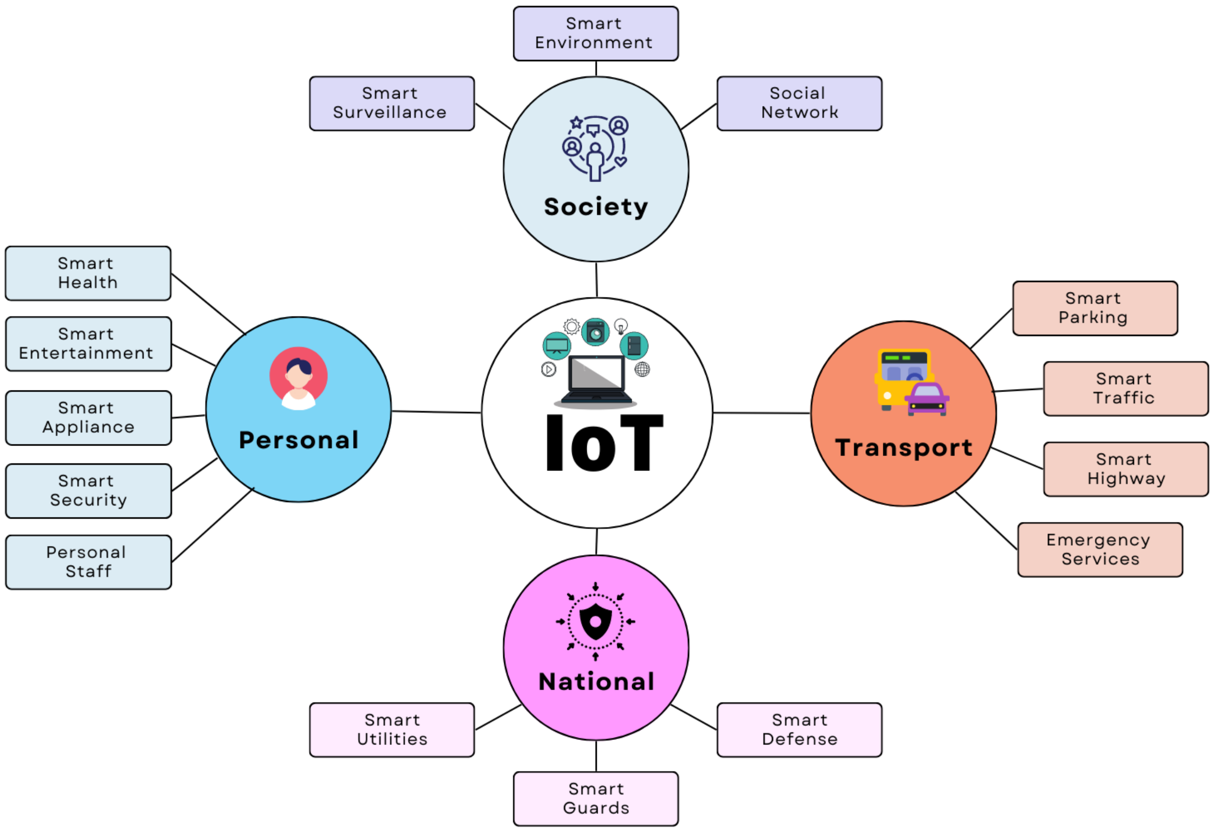


Figure 1.4: IoT components relevant to smart cities. (adapted from Whaiduzzaman et al., 2022)

There are several challenges facing the framework and infrastructure of the smart city that demand increased efforts from developers, scientists, and research institutes to provide answers and develop effective solutions.

The authors of (Bellini et al., 2022) state that there are interoperability problems due to smart cities have a wide range of IoT systems and devices from different suppliers and a wide number of IoT protocols, formats, and frameworks. This aspect is enhanced by the fact that many smart city applications have been initially developed as vertical silos, with each of them using its own solutions for data ingestion and storage. Interoperability challenges arise when these devices need to communicate and share data. Smart city communication networks are expanding in both size and complexity; this will not support interoperability between IoT nodes and clients. Therefore, there is no doubt that interoperability is one of the biggest challenges in IoT. Hence, a successful smart city should have effective measures to enrich interoperability amongst the existing single systems while conserving their operational independence (Al Mansoori, 2021).

Cybersecurity challenges are also important issues that government authorities and IT professionals consider to make real-time data secure and resilient against various forms of assault. Smart cities handle sensitive data related to citizens, infrastructure, and services. IoT protocols must be designed to address critical security and privacy concerns effectively; this includes secure authentication and authorization mechanisms, data encryption, secure firmware updates, and protection against various cyber threats. Encryption, confidentiality mechanisms, and access control measures can protect user privacy during data transfer.

One IoT platform can be used to collect and analyze all the data in the system; this could expose the system to many kinds of risks and vulnerabilities. Balancing security measures with efficient communication is essential to maintain optimal performance. There is always a risk that the huge amount of data being collected and communicated over the Internet of Things might end up in the hands of some misguided individuals. (Ilyas, 2023). One of the important performance challenges faced by IoT protocols in smart city environments is scalability; smart cities comprise a huge number of interconnected systems and devices that generate and consume massive amounts of data. IoT protocols must be able to meet the scalability requirements of smart city implementations. Guaranteeing effective communication between devices and the central infrastructure, enabling many thousands of devices, and managing growing data traffic.

Smart city applications, such as real-time monitoring, require low-latency communication. IoT protocols must minimize latency within the infrastructure and between devices to ensure smooth data exchange. Consistent network availability is another major requirement in the smart environment of the city to gather data and information from sensors and process them at high speed on the internet.

The massive data volumes generated by smart city networks raise QoS challenges. QoS sensitivity affects many smart city services and applications, including healthcare and intelligent grids. Enhancing power management for IoT devices and communication infrastructure in smart cities remains a challenge. The power consumption of these devices should be minimized. So many smart devices have low power consumption because they are battery-powered.

Another major issue is the government's lack of funds. An additional challenge in maintaining the smart city infrastructure is the shortage of skilled professionals. For communication protocols to support the connectivity of things and the wireless network, they must be low-latency, high data rate, scalable, high bandwidth, and long-range.

Table 1.1 outlines the key challenges commonly identified in the smart city domain, compiled and synthesized from a comprehensive review of relevant literature sources, including those indexed in Google Scholar, IEEE Xplore, Springer, MDPI, and the ACM Digital Library.

Table 1.1: Challenges in implementing a smart city

#	Challenges
1	Interoperability issues
2	Cybersecurity Issues and Data Privacy
3	Big data, large volume of data, data management
4	Cost, implementation challenges
5	High-power consumption
6	Operations and maintenance cost challenges
7	Low-latency tolerance
8	Affordability and scalability
9	Quality of service
10	Government funding - the government's insufficient

IMD publishes a yearly Smart City Index that combines economic and technological aspects of smart cities with "humane aspects" (quality of life and environment). The evaluation ranking of the city depends on several factors and structures such as safety, mobility, opportunities (Work & School), governance, affordable housing, road congestion, air pollution security, green spaces, health services, public transport, recycling, citizen engagement, basic amenities (water & waste), transparency, and school education. Table 1.2 presents the top ten smart cities listed in the 2024 Smart City Index.

Table 1.2 Top ten smart cities, listed in the 2024 Smart City Index. (IMD School, 2024)

#	Smart City
1	Zurich, Switzerland.
2	Oslo, Norway.
3	Canberra, Australia.
4	Geneva, Switzerland.
5	Singapore
6	Copenhagen, Denmark.
7	Lausanne, Switzerland.
8	London, England.
9	Helsinki, Finland.
10	Abu Dhabi, UAE.

1.2 Motivation

The way humans interact with their surroundings has been changed by the Internet of Things (IoT). The extraordinary increase in the number of internet-connected devices over the past two decades has attracted significant interest from engineers and academics in the field of the IoT (Schiller et al., 2022). According to Statista, the number of IoT devices is expected to increase from 10 billion in 2020 to approximately 30 billion in 2030, as shown in Figure 1.5. (Statista, 2023).

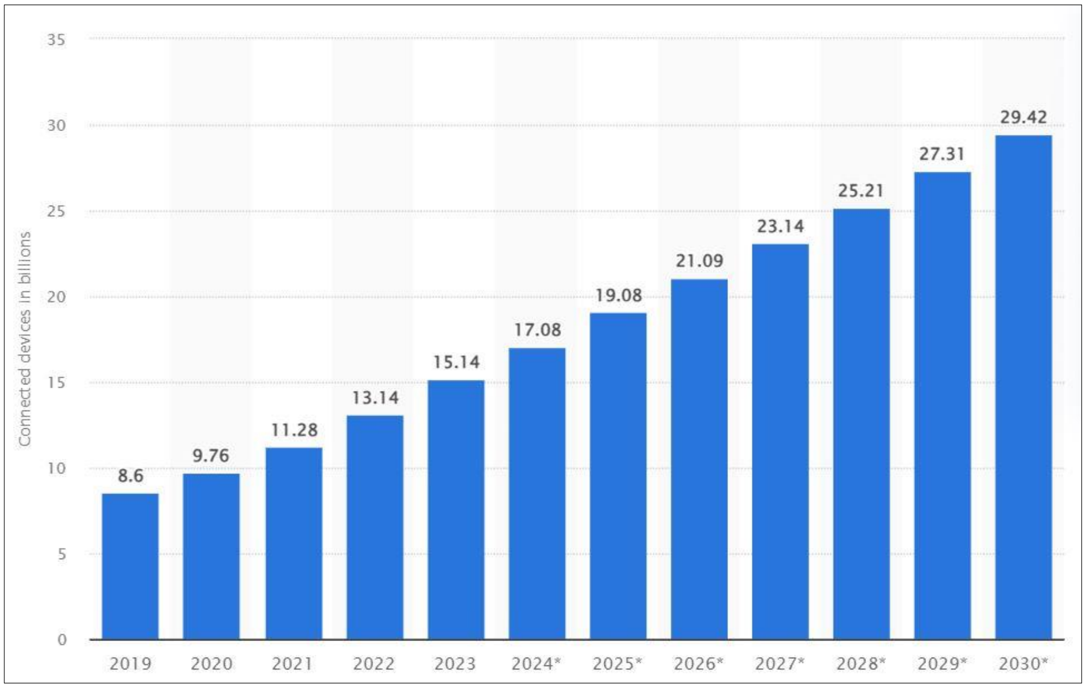


Figure 1.5. The number of devices that connected to IoT between 2019 and 2023. Number of devices expected to be connected to the Internet of Things by 2023

A primary goal of the Internet of Things (IoT) is to connect the physical and digital worlds. The IoT ecosystem is naturally diverse, comprising devices from many vendors and manufacturers with different hardware platforms, communication protocols, and data formats. This diversity creates challenges in achieving seamless interoperability and efficient device communication (Qiu et al., 2018), making it difficult to develop a universal standard that works for all devices. The variety of protocols and standards results in heterogeneity across IoT systems, which can occur from the design phase to choosing the right solution. Smart cities have attracted researchers' interest for the past twenty years. Using new technologies like IoT could help make smart cities more sustainable, efficient, and intelligent. This work aims to deepen the understanding of IoT protocols that can be used in smart city applications. The main concern for smart city developers and designers is which protocols will facilitate data exchange between nodes. So, the question remains: why should users choose one protocol over another?

1.3 Problem statement

Several IoT communication protocols are used to facilitate message exchanges between devices in IoT systems. Many messaging protocols have been developed to enable efficient communication within the IoT ecosystem, including HTTP, XMPP, MQTT, MQTT-SN, CoAP, AMQP, DDS, Matter, and WebSocket.

This thesis offers a comparative analysis of IoT messaging protocols, focusing on key features and insights from existing literature. Its main goal is to identify commonly used IoT communication protocols suitable for smart city development and to assess their performance. Figure 1.6 presents how IoT network protocols map onto the TCP/IP model. Therefore, understanding the characteristics and appropriateness of these protocols is crucial for choosing the best one for specific application scenarios. The protocols examined in this work are HTTP, MQTT, CoAP, AMQP, and XMPP.

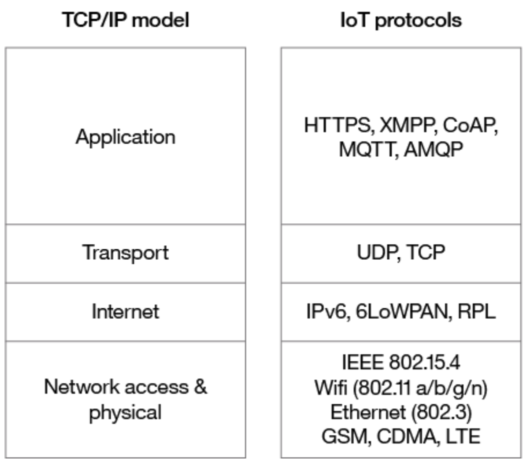


Figure 1.6 Protocols for IoT networks that are mapped to the TCP/IP model, adapted from (Gerber & Romeo, 2020)

1.4 Research Objectives

The primary goal is evaluating the efficiency of current IoT application layer protocols for smart city scenarios from a technological and scientific perspective and providing a better understanding of the answers to the following questions from the viewpoint of the developers of the IoT system, which are: 'What messaging protocols shall be used in the implementation of the IoT system in smart cities?' What are the differences in performance between one protocol and another? The protocols that have been the focus of this work are HTTP, MQTT, CoAP, AMQP, and XMPP.

Furthermore, this study aims to achieve these secondary objectives that address the specific needs related to IoT protocols and standards for smart cities:

- Investigate publications and institutions' websites for available IoT protocols and standards for smart city applications.
- Study of Internet-based protocols and emerging technologies for smart city applications.
- Comparison between available IoT protocols and standards for the smart city.
- Propose an Internet-based telecommunications protocol for general-purpose smart city applications.
- Comparison between the proposed IoT protocol and classic Internet protocols.

1.5 Thesis Contribution to the Field

This thesis investigates an important scientific contribution to the field of Internet of Things (IoT) protocols by examining their application in smart city environments. It aims to support researchers and developers working on building IoT systems within smart city services and to expand understanding of recent technologies related to the Internet of Things and smart city concepts.

The study presents a detailed comparative analysis of the five main IoT application layer protocols, including MQTT, HTTP, CoAP, AMQP, and XMPP, evaluated from a theoretical perspective. It provides researchers and engineers with a comprehensive understanding of application-layer protocols in terms of their performance. The analysis highlights their design philosophies, messaging models, and suitability for smart city applications with varying performance requirements. The thesis results can be used to help choose a messaging protocol and electronic platform and evaluate their performance. This motivated me to investigate the domain of IoT protocols more deeply. I aimed to identify the main challenges associated with IoT messaging protocols and their relationship to classical Internet protocols, examine the technical mechanisms involved behind the scenes, and propose solutions for academic and industry professionals.

1.6 Structure of the thesis

Chapter 1: This thesis begins with an introduction that outlines key concepts related to the Internet of Things (IoT), explains the motivation behind the study, defines the research objectives, and identifies the central problem to be addressed.

Chapter 2 contains the literature review.

Chapter 3 describes the proposed methodology and research design of the thesis, making a clear comparison between the protocols.

Chapter 4 describes the testbed that has been implemented and how the research method was applied, introducing both hardware setup and software setup for the experiments to evaluate the performance of the application layer protocols presented.

Chapter 5 reflects the experimental results of the tests and measurements that are demonstrated using diagrams and tables.

Chapter 6 presents the conclusion of the study and offers recommendations for future research.

Chapter 2: Literature Review

2.1 Introduction

The IoT is a challenging topic in both academia and industry. Nowadays, researchers have been concentrating a lot on IoT and have proposed multiple solutions. Since its appearance and the beginning of its development, IoT protocol performance has been studied from several levels.
A more comprehensive and deeper understanding of the subject is aimed to be provided. This chapter provides an overview and summary of the recent works related to this thesis topic, which has been widely researched in the last few years.

2.2 Literature Review

Application layer protocols for the IoT were investigated in (Mijovic et al., 2016). The performance of three application layer protocols was compared, including MQTT, CoAP, and WebSocket. These protocols have been implemented on the same hardware platform. The experiment evaluated the overhead and average round-trip time (RTT). Two different implementations have been considered. The first is a local area network (LAN) configuration, and the second is a more realistic IoT configuration, where the AP was connected to a remote server via the Internet. For the IoT configuration, two types of Internet connections are established: one using a home router and the second using a cellular network. From the results of their work, MQTT performance depends on the quality of service (QoS), and the CoAP protocol gained an excellent protocol efficiency and the lowest average RTT. Their study is relevant to this thesis as it involves a comparison of IoT protocols. While they employed an experimental approach to measure round-trip time (RTT) and overhead, our study focuses on calculating network latency.

(Bani Yassein et al., 2016) reviewed seven IoT application layer protocols, including XMPP, MQTT, CoAP, RESTFUL, DDS, AMQP, and WebSocket, in terms of architecture design, communication model, security, and QoS. They also reviewed the strengths and weaknesses of each protocol. In their study, they presented an extensive comparison of existing protocols and demonstrated that the selection of an appropriate application-layer protocol depends on various factors, including device capabilities, application requirements, and environmental conditions.

According to (Hantrakul et al., 2017), the study proposed a parking Lot software based on the MQTT protocol for data communication between devices. The proposed software can share real-time parking Lot information with any driver via a mobile phone. The experimental results indicate that HTTP consumes significantly more power than MQTT in dynamic data communication scenarios. Specifically, HTTP was found to use approximately ten times more power than MQTT. Moreover, during one hour of operation, MQTT successfully sent ten times more messages than HTTP."

Kayal, Paridhika, and Perros (2017) utilized a smart parking testbed to evaluate and compare the performance of CoAP, MQTT, XMPP, and WebSocket protocols. A smart parking application was implemented to compare the response time for different loads. The findings have shown that CoAP outperforms the other message queue-based protocols at lower server utilization. On the other hand, when considering that the program has the CPU power to support multithreading, WebSocket performs better than the other three protocols. XMPP performs better when the program supports multi-threading in order to minimize server utilization. Except for WebSocket, which shows the opposite tendency, the mean response time of all protocols increases as server occupancy increases.

A detailed examination of the four popular IoT protocols, HTTP, AMQP, CoAP, and MQTT, was presented by Naik (2017). Through a comparative analysis, his work summarizes the features of various protocols and provides insight into the strengths and limitations of these IoT protocols. Through the use of static components and empirical information from the literature, the author presents a comprehensive and comparative view of messaging protocols, enabling the end-user to make an informed decision about which protocol best suits their needs.

The study of (Khalil et al., 2018) investigated the general features of three protocols: CoAP, MQTT, and UPnP, to provide a subjective comparison from different perspectives. The authors also conducted performance analysis for the three protocols in terms of memory footprint, CPU footprint, latency, both upstream and downstream traffic, and power consumption. Although all three protocols have their respective advantages and disadvantages, the analytical evaluation indicates that CoAP outperforms both MQTT and UPnP in terms of performance. However, this comes at the cost of higher memory consumption. CoAP is also expected to become the de facto standard for resource discovery in IoT environments.

The authors in (Wukkadada et al., 2018) examined two application layer protocols; the first one is MQTT, and the other one is HTTP. They conducted experiments on the two protocols. The results show that MQTT packets have a 100% delivery rate and use less power.

The study by (Hofer & Pawaskar, 2018) analyzes two commonly used communication approaches in IoT environments. The benchmark compares the protocol MQTT against the RESTful approach, which is based on HTTP(s), in terms of energy consumption and performance. The results of this paper highlight that MQTT has strong advantages compared to REST due to the fact that these application layer protocols were specialized for IoT communication, while HTTP was extended to address new requirements. The proportion of data to overhead is better in MQTT, and the throughput is up to 3.85 times higher compared to REST. In addition, the QoS of MQTT can handle the reliability of the transmission in areas where the connection quality is poor.

The author of (Jaloudi, 2019) discussed the MQTT protocol used for event-based message communication in smart cities. His work focused on the evaluation of the MQTT protocol for IoT-based smart city applications. A network topology is developed that uses Wi-Fi as a communication infrastructure and is based on MQTT over TCP/IP. The "Raspberry Pi 3" platform is proposed as the MQTT server (broker). The inexpensive Wi-Fi module, which is an ESP8266, is used as one of the IoT-enabling and vital technologies. The module is a microcontroller-based system. In his study, the researcher implemented a protocol for completing practical scenarios at different levels of communication infrastructures, including sensor level, hub level, client level, and server level. In the proposed network model, JSON is used for formatting user data, and MQTT is used as a transfer protocol for packets. Based on the simulations conducted in the paper, the author concluded that the MQTT protocol is well-suited for small to medium-sized businesses utilizing IoT-based applications that communicate with online MQTT servers, as well as for medium to large-scale IoT applications that exchange messages through local MQTT servers. For the paper, some online servers (brokers) were tested, including the HiveMQ online testing server broker.hivemq.com, the Mosquito-based web server, which is available online as well on test.mosquitto.org, the test server introduced by Eclipse iot.eclipse.org, and the Moquette broker.moquette.io. The ping command is used to ping the hostname test.mosquitto.org, and the results show that the server broker HiveMQ has the lowest latency among them. From these results, it is concluded that the measurements do not depend only on the network latency but also on the server itself.

The contributors in (Marques et al., 2018) have proposed an architecture that contains a multilevel IoT-based smart city management system. They used a waste management problem as a use case to evaluate the performance of the proposed solution. In these scenarios, CoAP, HTTPS, and MQTT were used as IoT protocols to provide secure communication for the architecture implementation. The metrics used in the evaluation were energy consumption, latency, jitter, and throughput with different QoS. Results from MQTT were the best. Additionally, CoAP did very well. To optimize power consumption, the use of sleep mode is recommended.

The paper (Bansal & Kumar, 2019) begins by giving a practical overview of the concept of the Internet of Things, its layers, and components of IoT. The study also highlighted the challenges associated with smart cities, and then a comprehensive evaluation of application protocols MQTT, CoAP, and DDS is presented, and the utilization of these protocols in smart city applications is explained. The three metrics worked on were latency, bandwidth, and packet loss. Contiki software is used to analyze the performance of selected protocols. At low bandwidth, the MQTT protocol has a higher latency; as bandwidth increases, latency decreases, and the bandwidth used also decreases with increased packet loss. DDS has lower latency, but DDS consumes more bandwidth.

The research paper of (Sultana & Dumitrescu, 2019) evaluated commonly used application layer protocols using an IoVT (Internet of Video Things) framework for next-generation video surveillance systems. A real-time surveillance test application was implemented, and the communication protocols' performance was tested for parameters that affect these protocols, such as latency, throughput, bandwidth, overhead, memory usage, CPU usage, and energy consumption. The protocols for different video surveillance scenarios were ranked. The results demonstrate that the MQTT protocol can be applied and used to implement IoVT-based real-time video surveillance applications in a constrained environment. MQTT can transmit visual data from an IoVT edge node, as it uses less bandwidth and has suitable application latency. The large overhead and bandwidth consumption make the HTTP protocol inappropriate as an IoVT edge node protocol in a constrained environment, although it is the most widely used and adapted for different applications. HTTP was proposed by the author as an appropriate method for communication between a resourceful fog and an unconstrained cloud in an IoVT video surveillance case. For the CoAP protocol, a significant delay in large visual data being transmitted makes it inappropriate for an IoVT edge node." Due to its high overhead and latency in data transfer, XMPP is also not suitable for implementation in IoVT-edge nodes.

In (Zorkany et al., 2019), the study presented an e-health system, which is an automatic wireless health monitoring system used to measure patient parameters and utilizing Internet of Things technologies based on MQTT and CoAP protocols with embedded system technology. Based on these practical and simulation results, in terms of latency and the number of messages lost, the MQTT protocol provided better results than the CoAP protocol. The MQTT messaging protocol is well-suited for the design and implementation of an e-health platform within an IoT environment.

Smart grid systems benefit from IoT applications, which motivated the work in (Šikić et al., 2020). Three popular protocols, which are HTTP, MQTT, and AMQP, have been utilized to investigate the message performance in a smart grid integrated with IoT. The findings of the experiment demonstrate that the MQTT protocol is appropriate for all IoT-based smart grid use cases since it achieves the lowest message delivery time and the least amount of message overhead.

CoAP and MQTT protocols have been studied in (Kassem & Sleit, 2020). To assess performance, elapsed time is investigated. The work evaluated the impact of different simulation configurations on ECG signal processing, which was conducted in three different experiments. As the number of connected sensors increases, CoAP performs better. The results of the analysis help determine which of the IoT application protocols is used for ECG devices specifically and e-health and IoT devices in general.

(Seoane et al., 2021) have studied the performance of CoAP and MQTT, both the insecure and the secure versions, under different network conditions using a simple network scenario, contrasting in some simple situations with data obtained from an analytical model. The main conclusions of their study are that the MQTT protocol is more bandwidth demanding as it adds TCP overhead. On the other hand, it offers different QoS and implicitly offers reliability (for running over TCP), while CoAP offers a simple QoS and reliability mechanism through confirmable or non-confirmable messages. They also concluded that securing the communications through DTLS (CoAP) or TLS (MQTT) adds an important increase of the bandwidth usage – more than 1000% in CoAP and between 74 and just over 200% in MQTT – and the CPU usage – about 3.5% for PSK and 11.5% for PKI in CoAP and about 27% for PSK and 36% for PKI in MQTT – taking into account the modes of operation and QoS.

(Kumar & Jamwal, 2021) analyzed the protocols MQTT, XMPP, CoAP, CASTOR, AMQP, and LIDOR. These protocols have been studied and compared based on 6 parameters. The 6 parameters are communication overhead, security, packet loss, throughput, bandwidth, and support for QoS. To enhance clarity, the characteristics of each protocol are also outlined. By comparing these features, the differences between the protocols become evident, allowing the identification of the most efficient one based on the evaluated parameters. As a result, LIDOR was found to be the most effective communication protocol among those considered

It has been seen that LIDOR has an end-to-end feedback mechanism. The feedback mechanism helps LIDOR to reduce the overall communication overhead of the protocol. LIDOR enhances authenticity under DoS attacks by approximately 91%. Compared to CASTOR, LIDOR reduces network overhead by 32% and operates with low bandwidth requirements. It also provides strong support for Quality of Service (QoS). However, its main limitation is that it does not respond to packets that are lost once.

The work of (Silva et al., 2021) focuses on evaluating three of the most popular protocols used both in Consumer as well as in IoT environments for industrial applications. These protocols are MQTT, CoAP, and OPC UA. First, a local testbed for MQTT, COAP, and OPC UA has been carried out for experimentation. Then, larger experiments have now been carried out for MQTT and CoAP, based on the large-scale FIT-IoT testbed. Results show that OPC UA produced a higher time-to-completion compared to CoAP or MQTT. CoAP is the protocol with the lowest time-to-completion across all scenarios.

(Palmese et al., 2021) present a performance evaluation of two protocols: MQTT-SN, the version of MQTT for sensor networks, and CoAP in its Pub/Sub version. Both protocols are Pub/Sub in nature and based on UDP; therefore, they allow a fair comparison of their functionalities. The authors proposed an open-source implementation of the CoAP Publish/Subscribe model and compared it to MQTT-SN from both theoretical and practical perspectives within a simulated environment characterized by a varying number of clients and network conditions. The results indicate that CoAP is a viable alternative to MQTT-SN in publish-subscribe environments, and the authors suggest that CoAP is particularly well-suited for highly dynamic networks.

(Bayılmış et al., 2022) provide an overall review of IoT application layer communication protocols and challenges. They also discuss various technical issues to identify suitable IoT communication protocols, including REST, MQTT, CoAP, XMPP, DDS, AMQP, and WebSocket. The paper evaluates a range of relevant metrics, including the transmission model, message formats, security mechanisms, and QoS support. However, most IoT communication protocols, such as MQTT, CoAP, and WebSocket, have only been tested in limited environments, and their effectiveness in representative IoT scenarios has yet to be fully validated. The available results provide valuable insights to guide developers in selecting appropriate protocols.

When analyzing the experiment results, the CoAP protocol generally offers the best performance compared to other protocols. The results indicate that CoAP is suitable for IoT applications that require a high traffic load with low energy consumption and no demand for a handshake procedure. The main advantage of the MQTT protocol is its ability to efficiently transmit messages to multiple subscribers within the publisher/subscriber model. Therefore, the MQTT protocol may be preferred by IoT applications that require a secure communication environment and send messages to multiple subscribers at the same time. WebSocket is a good choice for IoT applications that desire high data rates. The paper also describes some challenges related to communication protocols for IoT applications and discusses some trends in different applications for IoT.

The project of (Tsvetanov & Pandurski, 2022) performs a real experiment on the XBee sensor network and the ThingSpeak cloud. HTTP, HTTPS, MQTT, and MQTT-SN protocols were used for transmitting the data packets between the sensor network and the cloud. The impact of the parameters of the transmitted packet on the latency has been studied. The results show that MQTT has various advantages over other protocols in terms of data throughput, CPU and RAM load, and integration between cloud structures and WSN sensor modules.

(Nwankwo et al. 2024) proposed an integration of the MQTT-CoAP protocol using an abstraction layer that enables both the MQTT-SN and CoAP protocols to be used in the same sensor node. Resources are managed by directly modifying the sensor node configuration using the CoAP protocol. Performance evaluation of these protocols under the integrated scenario shows acceptable levels of latency and energy consumption for Internet of Things (IoT) operations. The integration of the two protocols on a constrained device does not harm the system. The abstraction layer enables the simultaneous use of both MQTT-SN and CoAP protocols on resource-constrained IoT devices.

The paper by (Patti et al., 2024) proposes an IIoT-enabled version of MQTT called Prioritized MQTT (PrioMQTT) that can provide low latencies for time-critical messages. Unlike traditional MQTT, PrioMQTT is built on the UDP/IP protocol, providing enhanced support for latency-sensitive applications. It also features a system that prioritizes critical messages.

The combination of the UDP/IP stack and priority support in the PrioMQTT protocol is achieved while maintaining compliance with the MQTT standard message format. Therefore, PrioMQTT can be deployed on commercial-off-the-shelf (COTS) devices without hardware modifications.

The paper explains the PrioMQTT protocol and evaluates its performance in a realistic industrial setting, comparing it with the standard MQTT protocol. The results confirmed the validity of the proposed approach in reducing the RTT while properly handling the priorities of the messages.

According to a recent study (Mishra & Reddy, 2024), a comparison was introduced between MQTT, CoAP, and MQTTSN. The study evaluates the systems based on specific performance metrics, such as overhead, round-trip time, server response time, and reliability. Researchers establish ProtoLab as a realistic test bed and use it to assess the performance of protocols with different network configurations. According to their results, MQTT, CoAP, and MQTT SN performance vary based on the network conditions, such as static, dynamic, and poor network environments. In scenarios where the network remains steady or static network conditions, CoAP might demonstrate improved performance over MQTT and MQTT-SN under specific conditions. In dynamic network settings, MQTT-SN proves more effective than MQTT and CoAP, benefiting from its minimal overhead and design simplicity.

In poor network conditions marked by elevated latency, packet loss, or restricted bandwidth, MQTT can gain the upper hand thanks to its reliability features. The QoS levels offered by MQTT are equipped to manage scenarios with substantial packet loss, ensuring reliable message delivery even in demanding network conditions. The overall performance analysis indicates that MQTT-SN outperforms both MQTT and CoAP in supporting resource-constrained devices.

2.3 A summary of the research papers

A summary of the related research papers is presented in Table 2.1, along with an explanation of the performance metrics for each work.

Table 2.1: An overview of existing Papers on IoT application layer protocol.

#	work	Year	IoT protocol	Performance Metrics	Results show that
1	Mijovic et al.	2016	MQTT, CoAP, WebSocket	Overhead, average RTT.	The performance of MQTT is affected by the chosen Quality of Service (QoS) level, whereas CoAP demonstrated greater protocol efficiency and achieved the lowest average round-trip time (RTT) among the evaluated protocols.
2	Bani Yassein et al.	2016	XMPP, MQTT, CoAP, RESTFUL, DDS, AMQP, and WebSocket	security QoS.	Evaluated each protocol's benefits and drawbacks. The selection of a suitable application layer protocol depends on several factors.
3	Hantrakul et al.	2017	MQTT, HTTP	power consumption	The HTTP protocol consumed ten times more power than the MQTT protocol
4	Kayal & Perros.	2017	CoAP, MQTT, XMPP, WebSocket	response time.	CoAP outperforms the other message queue-based protocols when server utilization is low. On the other hand, when considering that the program has the CPU power to support

					multithreading, WebSocket performs better than the other three protocols. XMPP performs better when the program supports multi-threading in order to minimize server utilization. Except for WebSocket, which shows the opposite tendency, the average response time of all protocols increases as server occupancy increases.

5	Naik, 2017	2017	MQTT;
			CoAP;
			AMQP;
			HTTP;
				Overhead
				Bandwidth
				Latency
					To demonstrate a broader and comparative picture of messaging protocols, the author presents a comprehensive and comparative view of these protocols, enabling the end-user to make an informed decision about which protocol best suits their needs.

6	Khalil et al.	2018	CoAP, MQTT, and UPnP	memory & CPU footprint, latency.	CoAP outperforms both MQTT and UPnP, but it requires higher memory usage. CoAP is likely to be the de facto standard for resource discovery in IoT environments.
7	Wukkadada et al.	2018	HTTP, MQTT	power consumption	MQTT power consumption was lower than that of HTTP; messages sent by MQTT have a 100% delivery rate.
8	Hofer & Pawaskar	2018	HTTP, MQTT	Overhead, throughput. energy consumption	The proportion of data to overhead is better in MQTT, and the throughput is up to 3.85 times higher compared to REST. MQTT's Quality of Service (QoS) levels can ensure reliable message transmission, even in areas with poor connection quality.
9	Jaloudi.	2019	MQTT	latency over the cloud	MQTT protocol is suited for small to medium-sized IoT-based applications that exchange messages with online MQTT servers, and medium to large-scale applications that exchange messages with local MQTT servers.

					The measurements do not depend only on the network latency but also on the server itself.
10	Marques et al.	2018	CoAP, HTTPS, MQTT	latency, Jitter, throughput.	Results from MQTT were the best. Additionally, CoAP did very well. To optimize power consumption, enabling sleep mode is recommended.
11	Bansal & Kumar	2019	MQTT, CoAP, DDS	Latency, bandwidth, & Data loss	CoAP and DDS experience less latency as compared to MQTT.
12	Sultana & Dumitrescu	2019	MQTT, AMQP, HTTP, XMPP, CoAP, DDS.	latency, throughput, packet loss, bandwidth, overhead, energy	The MQTT protocol can be applied and used to implement IoVT-based real-time video surveillance applications in a constrained environment. MQTT can transmit visual data from an IoVT edge node, as it uses less bandwidth and has suitable application latency. The high overhead and bandwidth usage make the HTTP protocol unsuitable as an IoVT edge node protocol in constrained environments, even though it is the most widely used and adapted for various applications. The author proposed HTTP as a suitable communication method between a resourceful fog node and an unconstrained cloud in an IoVT video surveillance case." For the CoAP protocol, a significant delay in large visual data being transmitted makes it inappropriate for an IoVT edge node. Also, XMPP is not suitable for implementation in IoVT-edge nodes due to its high overhead and latency in data transfer, XMPP is not suitable for implementation in IoVT-edge nodes

13	Zorkany et al.	2019	MQTT, CoAP	average byte, ratio delay.	The MQTT protocol provided better results than the CoAP protocol in terms of delay, the number of messages lost.
14	Šikić et al.	2020	HTTP, MQTT, AMQP	traffic load and delivery latency.	The MQTT protocol is appropriate for all IoT-based smart grid use cases since it achieves the lowest message delivery time and the smallest protocol overhead.
15	Kassem & Sleit.	2020	MQTT, CoAP	elapsed time.	When the sensor network size increases, CoAP continues to show better performance.
16	Seoane et al.	2021	MQTT, CoAP	bandwidth & CPU use	The MQTT protocol is more bandwidth demanding as it adds TCP overhead, but it offers different QoS.
17	Kumar & Jamwal	2021	MQTT, XMPP, CoAP, CASTOR, AMQP, and LIDOR	Overhead Security packet loss bandwidth QoS	Concluding that LIDOR is the most efficient due to its end-to-end feedback mechanism, high QoS support, and strong performance under DoS attacks.
18	Silva et al.	2021	MQTT, CoAP, and OPC UA	time-to-completion and packet loss.	OPC UA, produced a higher time-to-completion in comparison to CoAP or MQTT. CoAP is the protocol with the lowest time-to-completion across all scenarios.
19	Palmese et al.	2021	CoAP vs. MQTT-SN	End-to-end delay.	CoAP is the best choice for highly dynamic networks.
20	Bayılmış et al.	2022	MQTT, CoAP, XMPP, DDS, AMQP, and WebSocket,	Throughput, delay time, energy consumption	CoAP is well-suited for IoT applications that involve high traffic loads, require low energy consumption, and do not demand a handshake procedure. MQTT is well-suited for IoT scenarios that require secure data exchange and the simultaneous delivery of messages to multiple subscribers. WebSocket is a good choice for IoT applications that desire high data rates.

21	Tsvetanov & Pandurski,	2022	HTTP, HTTPS, MQTT, and MQTT-SN,	data rate, CPU, and RAM load	The results give some advantages of MQTT over other protocols in terms of data rate, CPU, and RAM load when working with XBee sensor modules and integration between WSN and cloud structures.
22	Nwankwo et al.	2024	MQTT-SN, CoAP	latency and energy consumption	The abstraction layer provides the ability for both MQTT-SN and CoAP protocols to be used simultaneously on constrained IoT devices.
23	Patti et al.	2024	MQTT	latency	proposed an IIoT-enabled version of MQTT called a Prioritized MQTT (PrioMQTT), which reduces the RTT while properly handling the priorities of the messages.
24	Mishra & Reddy.	2024	MQTT, CoAP, and MQTTSN	overhead, RTT, server response time reliability.	The results show that the performance of MQTT, CoAP, and MQTTSN can differ significantly depending on the network conditions. Performance comparison shows that the MQTTSN is better than the MQTT and CoAP protocols for constrained devices.
25	Tran et al.	2024	MQTT, HTTP, AMQP	bandwidth, latency.	MQTT and AMQP play a role in enhancing overall efficiency and speed within the framework of our suggested photovoltaic system.

2.4 Discussion for literature review

Table 2.1 summarizes the key concepts of the related works. It shows the year the paper was published, the IoT protocol used in each research paper, and the performance metrics for each work, where a variety of metrics were used in these papers to assess and compare the performance, including the amount of data sent, overhead, latency, round-trip time (RTT), bandwidth, Packet Loss, and throughput. A network's performance can be determined by several factors, including the number of users, the type of transmission media, the capabilities of the connected devices, and the software's efficiency.

It was also noticed that the most quantitative metric that was measured to indicate the tested protocol's performance was the message latency. In general, these metrics can be divided into categories depicted in Table 2.2.

A. Time-Based

The first category includes time-related metrics such as latency, jitter, round-trip time, throughput, and response time, which means that the time required to transfer data packets between IoT endpoints or nodes. Many papers discussed and evaluated these important metrics. For example, the study (Šikić et al., 2020), showed that the MQTT protocol is appropriate for all IoT-based smart grid use cases since it achieves the lowest message delivery time and the minimal amount of data overhead. Another study's results showed that MQTT is four times faster than the HTTP protocol when comparing the message sent latencies. This is because MQTT is a message-oriented protocol that uses the publish-subscribe approach. In contrast, the HTTP protocol uses the request-response model, which is a document-oriented protocol.

According to the study in (Jaloudi, 2019), online brokers were tested, and the results show that the server broker HiveMQ has the lowest latency among them. From these results, it is concluded that the measurements not only depend on the network latency but also on the server itself.

The results of (Sultana & Dumitrescu, 2019) showed that MQTT can transmit visual data from an IoVT edge node, as it uses less bandwidth and has suitable application latency in a constrained environment. A comparison between CoAP and MQTT was conducted in the experimental work (Zorkany et al., 2019). From the practical implementation of these protocols, the MQTT protocol revealed better results than the CoAP protocol in terms of latency, the number of messages lost, and the number of bytes used in messages. The fact is that implementing an appropriate messaging protocol reduces latency through IoT nodes.

B. Packet and Transmission

The second category includes packets and their relevant features for data transfer, such as payload and overhead. For HTTP, due to its high bandwidth consumption and significant data overhead, the protocol is considered unsuitable for use as an IoVT (Internet of Video Things) edge node protocol in resource-constrained environments. Although CoAP consumes less bandwidth and incurs reduced costs per packet, the high latency when sending large amounts of data makes it unsuitable for IoVT edge nodes. Additionally, the XMPP Protocol did not seem suitable for this scenario due to its high data transfer latency (Hofer & Pawaskar, 2018). As a result of the study, MQTT has significant advantages over REST HTTP because these application layer protocols are specialized for IoT communication, while HTTP has been extended to meet new requirements. The data-to-overhead ratio is better in MQTT, and its throughput can be up to 3.85 times higher than REST HTTP. It is important to remember that large headers can cause performance issues, increased latency, or even denial-of-service attacks. Therefore, it is recommended to keep HTTP headers at a suitable length. CoAP is the most appropriate protocol for applications with constrained resources, as it introduces the least overhead, according to a study (Mishra & Reddy, 2024). A study by Khalil et al. (2018) mentioned that CoAP uses a binary format for encoding, with a header size of 4 bytes. For MQTT, the payload message has a maximum size of 256 MB, with a message header length of 2 bytes for each MQTT command. CoAP performs better on low-power and network-constrained devices. While MQTT is the best choice for applications that rely on the pub/sub methodology, after reviewing every paper that has been evaluated, it is clear that, as of right now, there are no common or standard technologies or protocols that can be employed with an Internet of Things system. XMPP has a larger overhead as it embeds messages in XML stanzas.

C. Energy and Power Use

The third category is concerned with energy and power use, as the study results by Wukkadada et al. (2018) show that MQTT power consumption is lower than that of HTTP and also indicate that the messages sent by MQTT have a 100% delivery rate. This is what the study (Hantrakul et al., 2017) confirmed: HTTP is observed to be consuming more power. In the tests conducted, the HTTP protocol consumed 10 times more power than the MQTT protocol, and the fourth category focuses on hardware usage, including CPU and RAM usage.

Based on the related works discussed in Table 2.1, it was found that the MQTT performance significantly depends on the Quality of Service (QoS), as mentioned in [(Mijovic et al., 2016), (Marques et al., 2018)]. This QoS is important in MQTT due to its role in providing the client with the ability to select a service level that corresponds with the application's requirements.

These studies used a variety of hardware platforms and software platforms, such as STMicroelectronics Nucleo-F411RE, as in paper (Mijovic et al., 2016), the Raspberry Pi as in papers (Khalil et al., 2018), (Hofer & Pawaskar, 2018), (Jaloudi, 2019), (Marques et al., 2018), (Sultana & Dumitrescu, 2019), ESP8266 as in (Wukkadada et al., 2018), (Jaloudi, 2019), (Marques et al., 2018). Arduino as in (Sultana & Dumitrescu, 2019) and simulation using Contiki software as in (Bansal & Kumar, 2019), (Šikić et al., 2020), or simulation using WANEM software as in (Zorkany et al., 2019), or Network SIM-IOTIFY as in (Šikić et al., 2020), as well as technologies such as wired and wireless networks. As a result, it is concluded that there is no general model or approach to investigate the performance comparison.

The protocol that uses the least power is MQTT, and it is suitable for energy-constrained IoT environments (Dash & Peng, 2022). Therefore, it is used in scenarios that enable reducing battery power consumption in smart cities.

Table 2.2: The three categories the literature review focuses on

	Category	Relevant features
1	Time-Based	Round Trip Time (RTT), delay(latency), jitter & throughput.
2	Packet and Transmission	Payload, Message Delivery Rate, and Overhead.
3	Energy	Battery Life, Energy, Power use.

Chapter 3: Methodology 1

Comparative study of main IoT protocols

3.1 Introduction

One of the important tasks carried out by the communication protocol in the architecture of the IoT is to collect data from nodes, send it, and store it in the appropriate form for the application scenario. Each layer of the IoT architecture has a set of protocols. In the following section, the author reviews some of the common protocols in the application layer. These protocols are HTTP, MQTT, CoAP, AMQP, and XMPP. These protocols will be introduced and conceptually compared.

3.2 Application-Layer Protocols Overview

3.2.1 Hypertext Transfer Protocol (HTTP)

HTTP is a web messaging protocol used for data exchange. It is a client-server protocol, based on the request-response RESTful Web architecture model (IETF Datatracker, 2022). Requests are sent from the client to the server, and a response is sent back from the server. HTTP utilizes Universal Resource Identifiers (URIs) and operates as a text-based protocol that represents all data in human-readable ASCII text format. It is a text-based protocol that expresses all data as human-readable ASCII text. Besides the actual data, specific meta information is transmitted in both the HTTP request and the server's response, which is located inside the HTTP header.

HTTP itself depends on TCP/IP to get requests and responses between the client and server. By default, TCP port 80 is used, but other ports can also be used. HTTPS, however, uses port 443. HTTP data is carried above the TCP protocol, ensuring reliability of delivery, and these are the steps by which the HTTP protocol transmits data packets, as shown in Figure 3.1:

In the HTTP communication process, the client initiates a connection by sending a SYN message to the server. Then, the server responds with a SYN-ACK packet, to which the client replies with an ACK packet, completing the connection establishment. This process is commonly referred to as a three-way handshake. After the connection is established, the client sends an HTTP request to the server to access a specific resource. The client then waits for the server to process the request. The web server executes the request, locates the desired resource, and returns the appropriate response to the client. When the client no longer requires additional resources, it terminates the TCP connection by sending a FIN packet. Because HTTP uses familiar and compatible web technologies. This advantage makes this protocol one of the most important protocols used for the Internet of Things, and its integration with web services is simple and efficient.

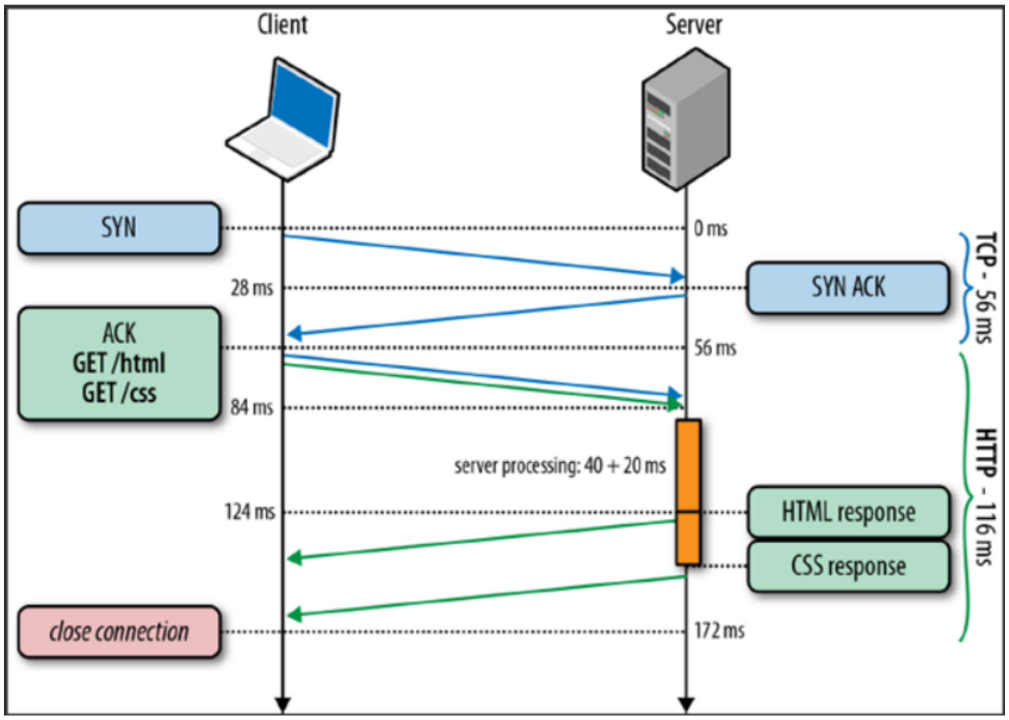


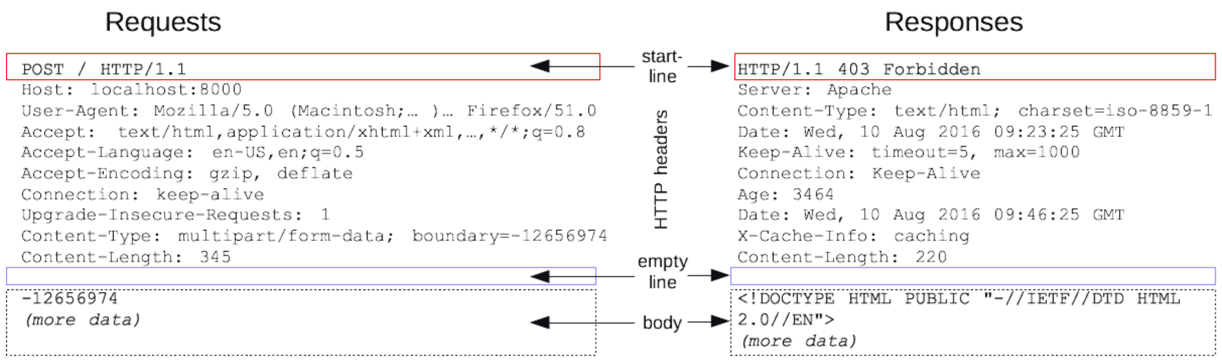
Figure 3.1: HTTP Three-way handshake in opening a connection

The four most common methods used in it are GET, POST, PUT, and DELETE. GET is used to obtain data from a server, POST is used to send data to a server, PUT is used to update existing data on a server, and DELETE is used to remove data from a server. These are the main methods of RESTful applications and services. Additionally, the server provides various types of responses, called response status codes, and they are represented by numbers. These codes indicate whether a specific HTTP request has been completed (Mozilla, 2023).

HTTP requests and responses share a similar structure and are composed of (Mozilla, 2023):

1. A start-line describing the requests to be implemented, or their status, whether successful or a failure. This is always a single line.
2. An optional set of HTTP headers specifying the request, or describing the body included in the message.
3. A blank line indicates that all meta-information for the request has been sent.
4. An optional body containing data associated with the request (like the content of an HTML form), or the document associated with a response. The presence of the body and its size are specified by the start-line and HTTP headers.

The start-line and HTTP headers of the HTTP message are collectively known as the head of the request, whereas its payload is known as the body, as shown in Figure 3.2.



```
POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh;… )… Firefox/51.0
Accept: text/html,application/xhtml+xml,…,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content-Length: 345

-12656974
(more data)
```

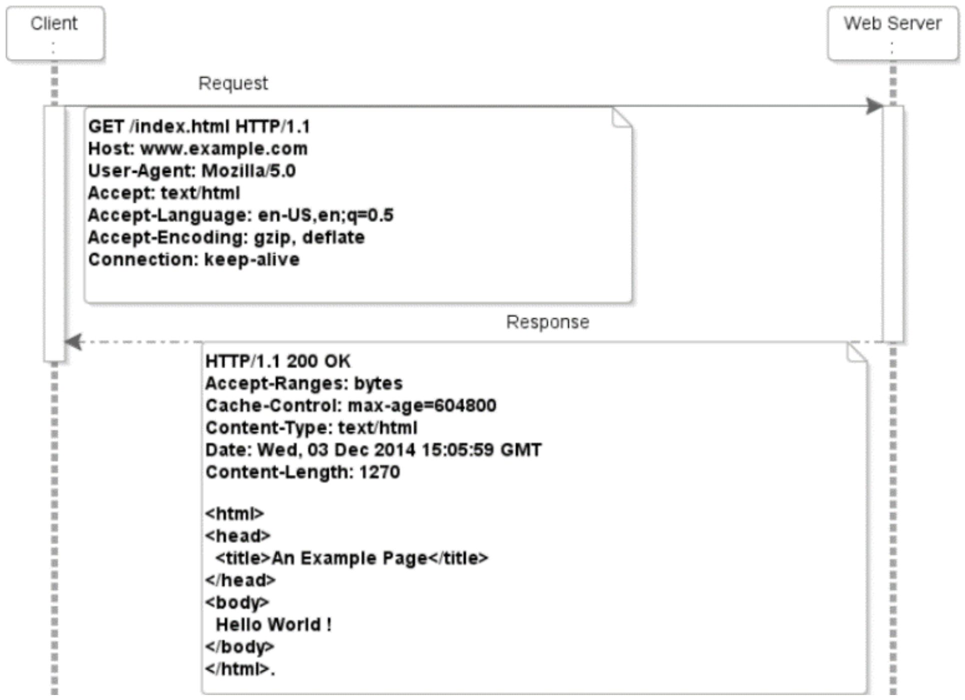
```
HTTP/1.1 403 Forbidden
Server: Apache
Content-Type: text/html; charset=iso-8859-1
Date: Wed, 10 Aug 2016 09:23:25 GMT
Keep-Alive: timeout=5, max=1000
Connection: Keep-Alive
Age: 3464
Date: Wed, 10 Aug 2016 09:46:25 GMT
X-Cache-Info: caching
Content-Length: 220

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML
2.0//EN">
(more data)
```

Figure 3.2: An example of an HTTP request and response message

In an IoT environment, a common use of HTTP is to allow devices to POST to a resource that represents the device state on the IoT service (HiveMQ, 2023).

HTTP has recently been connected to the REST architecture to enable communication between different components through web services. Various data formats, including JSON and XML, are supported, making it suitable for diverse IoT data exchange requirements. A huge number of tiny packets need to be transmitted via HTTP, which is not designed for IoT applications (IETF Datatracker, 2022).



```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0
Accept: text/html
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Cache-Control: max-age=604800
Content-Type: text/html
Date: Wed, 03 Dec 2014 15:05:59 GMT
Content-Length: 1270

<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World !
</body>
</html>.
```

Figure 3.3: basic request-response model of communication between a web browser and a server over HTTP.

3.2.2 Message Queuing Telemetry Transport (MQTT)

MQTT is a lightweight, event-based protocol. Based on the client-server paradigm, it is a message-oriented protocol; nevertheless, the server that runs it functions more like a broker or gateway (MQTT, 2023). It was invented by IBM, aiming to reduce bandwidth requirements, and is a suitable choice for IoT devices. It is a great protocol for connecting remote devices using the least amount of network resources. The communication between the client and broker is connection-oriented because it is built on top of TCP. See Figure 3.4

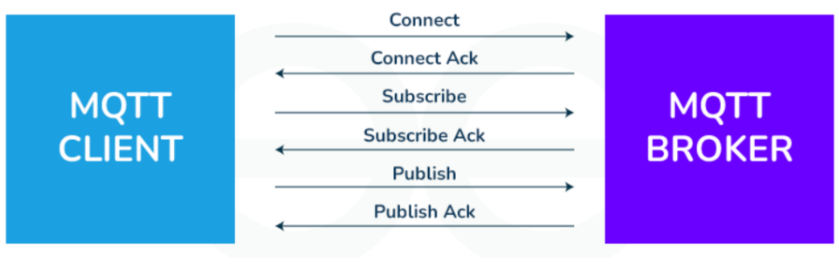


Figure 3.4: The message request-response in the MQTT protocol

MQTT follows the publish-subscribe paradigm, which is defined by three actors: the publisher, the subscriber, and the broker, as shown in Figure 3.5. The publisher creates the data and sends messages to the broker to publish the data. The broker then sends the data to all subscribers who have subscribed to that data topic. In the publish-subscribe pattern architecture, each node is either a data producer (publisher) or a data consumer (subscriber).

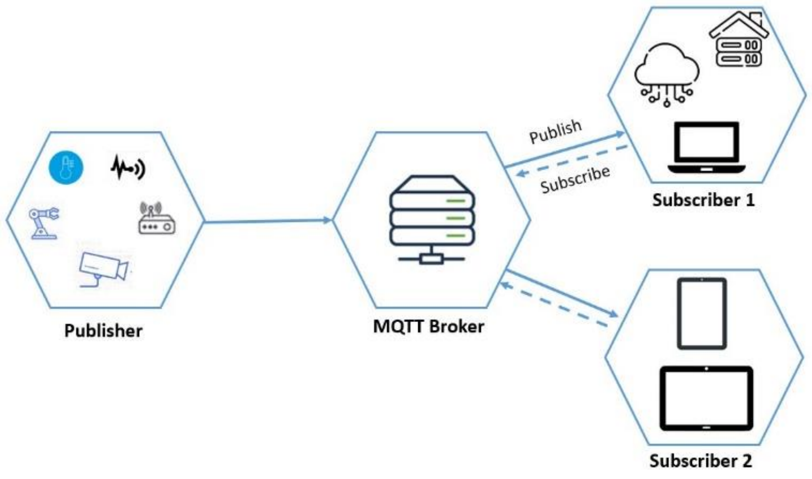


Figure 3.5: MQTT architecture.

MQTT is characterized as a binary-based protocol, in which control elements are represented by binary bytes rather than text strings (Steve's-internet-guide, 2023). Then the data is exchanged as plain text. Since MQTT runs on top of TCP, communication between client and broker is connection-oriented. The broker handles messages for each device connection using a topic. Generally, a topic is composed of multiple levels separated by a slash.

One of the significant features of this protocol is that the subscriber does not need to know the publisher's IP; on the other hand, the broker needs to be recognized by each client's MQTT protocol port and IP address.

Within the context of Quality of Service (QoS), levels are agreements between the system nodes that define the guarantee of delivery for specific messages. MQTT utilizes several quality service levels that provide different delivery guarantees for messages. Both the subscribe and publish message bodies can be defined as one of these levels, so MQTT delivers messages using three QoS levels. QoS 0, 1, and 2 (MQTT, 2023):

QoS level 0: (At most once), a message is sent only once without checking whether it reaches its destination, so the messages are not guaranteed to arrive. Loss may occur without acknowledgement and confirmation of message reception. QoS level 0 is often referred to as "fire and forget."

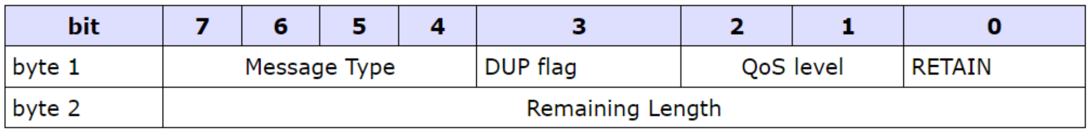
QoS level 1 (At least once), messages are guaranteed to be delivered to the recipient at least once, but duplicates may be received if acknowledgments are lost or retransmissions occur. This QoS level ensures reliable message delivery through acknowledgment packets; however, the receiving application is responsible for handling potential message duplication.

QoS level 2 (Exactly once), messages are guaranteed to be delivered to the intended recipient exactly once, with no duplication or loss. This level employs a four-step handshake process with acknowledgments to ensure message integrity and uniqueness. While QoS 2 offers the highest level of reliability among MQTT quality-of-service levels, it is associated with increased latency and protocol overhead due to its more complex delivery mechanism. Therefore, selecting an appropriate QoS level should be carefully aligned with specific network conditions and application requirements. By utilizing QoS levels 1 and 2, reliable message delivery can be achieved, ensuring that data packets reach their destination even under unstable network conditions.

MQTT's scalability and reliability are among its main benefits (Spohn, 2022). The MQTT protocol is utilized by each of the leading cloud systems, including Microsoft Azure, IBM Cloud, and Amazon Web Services. It also gets used by both the Messenger and Instagram apps on Facebook and is considered the most favorable connection protocol for M2M and IoT (Bani Yassein et al. 2016).

Message Payload

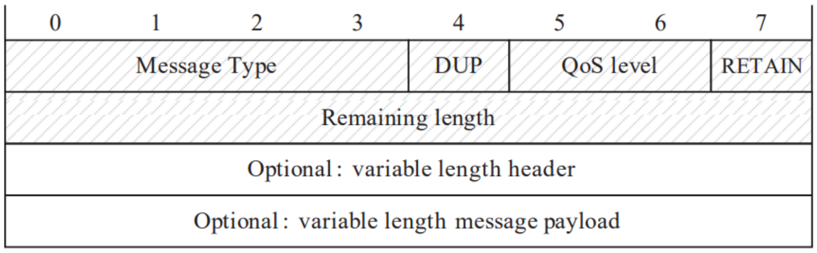
A 256MB maximum MQTT PUBLISH payload size is specified by the MQTT protocol; however, most clients in M2M and IoT scenarios are unlikely to send messages with such large payload. The message header for each MQTT command message typically contains a fixed header, which is 2 bytes long, as shown in Figure 3.6.



bit	7	6	5	4	3	2	1	0
byte 1	Message Type				DUP flag	QoS level		RETAIN
byte 2	Remaining Length							

Figure 3.6: MQTT Fixed Header Format.

Some messages also require a variable header that contains protocol name, protocol version, topic name, and several flags, and a payload message, which is the actual data being transmitted. Figure 3.7 shows the message format defined by the MQTT protocol, which includes a fixed header, an optional variable header, and a payload.



0	1	2	3	4	5	6	7
Message Type				DUP	QoS level		RETAIN
Remaining length							
Optional : variable length header							
Optional : variable length message payload							

Figure 3.7: IoT MQTT Message Format

Regarding MQTT data transmission, MQTT transmits messages over a TCP connection. The Eclipse Foundation developed and maintains the Java-based Paho library, which was applied to implement the client-side application [eclipse]. The Paho library provides fully developed implementations that enable a developer to send data as MQTT messages over TCP and WebSocket connections. Paho library has these features; there's no need for a self-developed solution to complete the work. MQTT is used in many industries today, including remote monitoring, automotive, forestry, messaging applications, home automation, and more. In addition, it is beneficial and suitable for Wireless Sensor Network applications.

3.2.3 Constrained Application Protocol (CoAP)

The Constrained RESTful Environments (CoRE) working group of the Internet Engineering Task Force (IETF) developed the CoAP protocol in June 2014. The standard for this protocol is described in the document RFC 7252 (IETF Datatracker, 2022). CoAP follows a client-server architecture, where clients send requests to servers, which in turn respond with the requested data. CoAP and HTTP both have the same methods, like GET, PUT, POST, and DELETE.

It employs UDP packets for transmission, primarily designed for small, low-power, and constrained devices, to support REST services in M2M communication, as well as for communicating with HTTP using simple proxies, thereby supporting constrained devices and networks.

The packet size of CoAP is smaller than that of HTTP TCP packets. CoAP defines four different types of messages: confirmable, non-confirmable, acknowledgement, and reset. (Eggly et al., 2018).

A. Confirmable (CON). These messages require an acknowledgement to be sent by the other communicating part. When the network does not cause packet losses, each CON message triggers exactly one return message of type Acknowledgement or type Reset. If no ACK or RST is received, after a certain time, the CON message is assumed to be lost, and it is retransmitted.
B. Non-confirmable (NON). These messages do not require an acknowledgement, offering no reliability.
C. Acknowledgement (ACK). An ACK message acknowledges receipt of a particular CON message. It is also able to carry the response to the request, a process known as piggybacked response.
D. Reset (RST). This message reports that a particular message (CON or NON) was received, but it cannot be properly processed. This event typically occurs when the receiver has rebooted and forgotten some state required to interpret the message correctly. Provoking a Reset message (e.g., by sending an Empty CON message) is also useful for checking the liveness of an endpoint, such as with a CoAP ping.

The CoAP protocol is commonly found in smart home environments, as well as in traffic monitoring and environmental monitoring applications. CoAP can connect multiple IoT devices, such as streetlights, garbage cans, and parking sensors, in smart city projects.

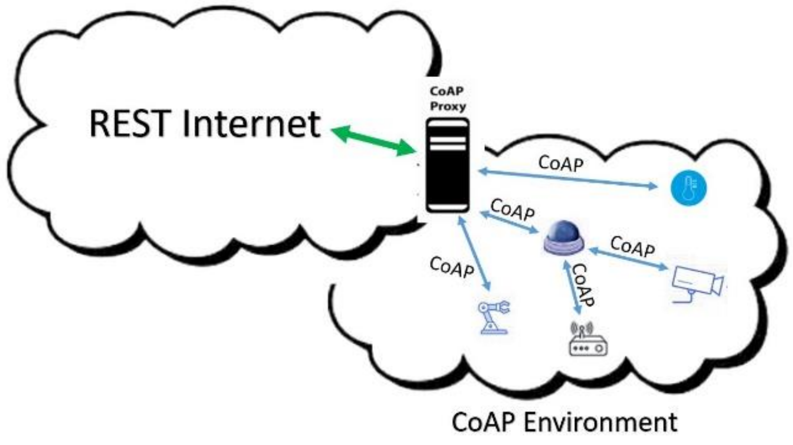
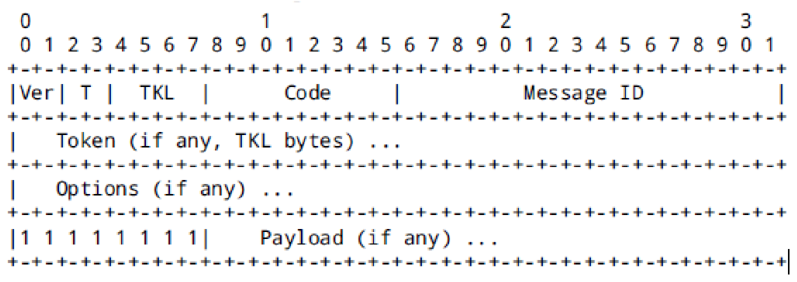


Figure 3.8: CoAP architecture.

To reduce the typical overhead caused by protocol headers, the message format in the protocol has been designed to be light and straightforward. Simple binary encoding is used for CoAP messages. The CoAP message format consists of a fixed 4-byte header, followed by a code field that ranges from 0 to 8 bytes long. As shown in Figure 3.9, a one-byte payload marker (0xFF) appears before the payload field if it exists. Only the 4-byte header is required; the rest is optional.



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver| T |  TKL  |      Code     |          Message ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Token (if any, TKL bytes) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+|
```

Figure 3.9: IoT CoAP Message Format adapted from (Becker, 2013).

3.2.4 Extensible Messaging and Presence Protocol (XMPP)

XMPP is widely recognized as a popular protocol for instant messaging and has been adopted by various Internet platforms, including Google Talk, Cisco, Live Journal, and BBC Radio Live Text, among others. It is an open-source protocol used for applications involving instant messaging for real-time data streaming, such as audio and video communication.

XMPP is one of the protocols that supports the publish-subscribe pattern (XMPP, 2023) that uses a client-server architecture to deliver the message, but also XMPP supports request/ response architecture, and the choice of which architecture to use is on the application developer. It also provides server-to-server communication where communication between two clients is not possible. "XML stanzas," which are XML-based messages, are used for communication between the client and the server, creating additional loads as a result of unnecessary tags, and they also require an XML parser, which requires computing capacity that increases energy consumption. The network communication is based on a client-server approach. Figure 3.10 illustrates the overall behavior of the XMPP protocol, among which gateways can bridge between foreign messaging networks. XMPP creates a unique ID called Jabber ID, which gives the information of the client to whom a message is to be sent. This ID is used by the XMPP server to route the message across a TCP connection. An XML server stream is established with the server, and the client is identified using a Jabber ID. The server will begin another XML stream after determining the client. The server will send the stream to the client, making the XML stream bidirectional. One of the weak points of the protocol is that data transfer is comparatively slow, as XMPP uses a thin bath for transferring binary data. Another weak point must have been mentioned here: the XML message added additional overhead because it has a lot of headers and tag formats, so it consumes more power.

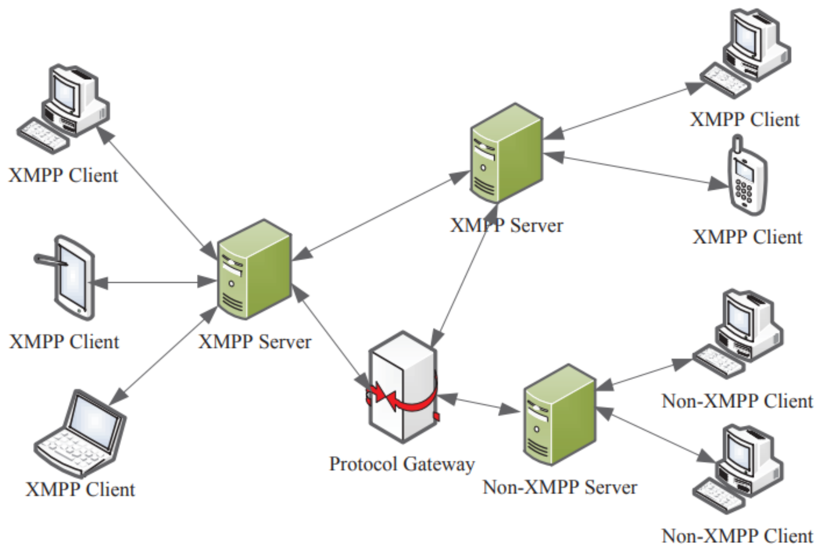


Figure 3.10: XMPP communication system architecture adapted from (Wang et al., 2017).

3.2.5 Advanced Message Queuing Protocol (AMQP)

AMQP is an application layer protocol standardized by OASIS and designed for message-oriented networks & it has support for Publisher/Subscriber architecture. It uses TCP as a transport protocol to provide reliable communication. Besides this, to provide QoS guarantees, it has three levels of delivery, namely at least once, at most once & exactly once. Along with Publishers & Subscribers, it has two more components, Exchanges & Message queues. Exchanges perform the routing functionality by forwarding messages to appropriate message queues. These messages can be stored in message queues before forwarding them to Subscribers. AMQP uses two different message types. First, bare messages which are used by Publishers, and second, annotated messages which are used by Subscribers (Al-Fuqaha et al., 2015).

Advanced Message Queue Protocol (AMQP) originated in the financial services industry in 2006. It was initially designed for financial transaction processing systems, such as trading and banking systems, which require high guarantees of reliability, scalability, and manageability. It is an increasingly important protocol for message-oriented middleware (MOM) with its origin in the financial services industry. The main purpose of the AMQP protocol is to handle thousands of queued transactions. AMQP is a middleware protocol extensively used for exchanging messages in distributed applications. It is an asynchronous message queuing protocol that aims to create an open standard for transmitting messages between applications and systems regardless of internal design. It was designed to enable interoperability between different applications and systems.

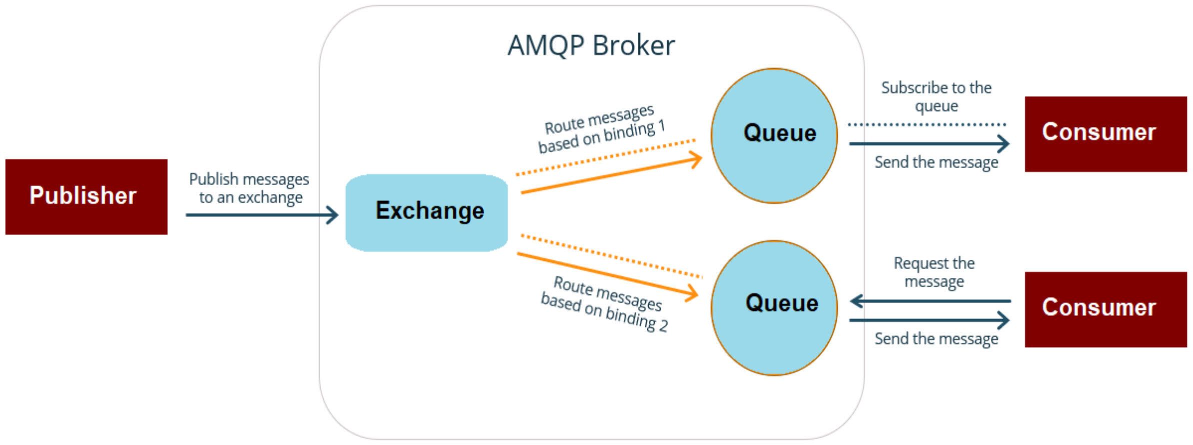


Figure 3.11: AMQP communication architecture

3.3 Chapter discussion

Over the past two decades, several studies have been published that compare the performance of IoT application protocols. In this section, the features of the protocols will be compared on a conceptual level using evidence from the literature to make a clear comparison between the protocols. The characteristics of different application layer protocols are shown in Table 3.1

Table 3.1: Characteristics of application layer protocols MQTT, CoAP, XMPP, AMQP, and HTTP

Protocol	MQTT	CoAP	HTTP	XMPP	AMQP
Main Purpose	M2M, Constrained devices	low-overhead M2M, Constrained devices	web messaging protocol.	Instant messaging, XML data	M2M
Methodology	Message oriented (Data-centric)	Document oriented	Document oriented	Message oriented	message oriented
Architecture	broker/ client	Client/ Server Client/ Broker	Client/ Server	Client/ Server	broker/ client
Pattern	Publish/ Subscribe	Publish/ Subscribe Request/ Response	Request/ Response	Publish/Subscribe	Publish/ Subscribe Request/ Response
Transport Layer	TCP	UDP	TCP	TCP	TCP
Header Size	2 byte	4 byte	Undefined	Undefined	8 byte
Payload Format	Supports binary and text payloads	Supports binary and text payloads	Text, Binary in HTML 2	XML	binary and text payloads
Message Size	256 MB maximum	Small & undefined	Undefined	64 KB stanza	128 MB max recommended

			Large	size	
Security	TLS/SSL	DTLS, SSL	TLS/SSL	TLS/SSL	TLS/SSL
Default Port	1883,8883 (TLS/SSL)	5683 (UDP)/ 5684 (DLTS)	80 and 443	5222 and 5269 secure (TLS) 5223.	5672 or 5671
QoS	• Exactly once • At least once • At most once	Confirmable Non-Confirmable	not support	not support	• Exactly once • At least once • At most once
Licensing	Open Source	Open Source	Free	Open Source	Open Source
Standard	OASIS	IETF	IETF	IETF	OASIS

Studying the differences between available IoT protocols will help in selecting the best option for an IoT scenario. The selection between these protocols should be made after considering several features and characteristics such as network architecture, Methodology, messaging pattern, Transport Layer, payload size and format, security mechanisms, and quality of service options.

MQTT is designed for M2M communications in constrained networks, and CoAP was developed for low-overhead M2M constrained devices. AMQP is a lightweight M2M protocol, and XMPP is an Instant Messaging (IM) protocol developed essentially to allow communication between people through messages. HTTP was originally designed for the Web and not for the IoT; therefore, it requires maximum overhead and a larger message size among all protocols.

For connectivity issues, HTTP, MQTT, AMQP, and XMPP are designed to run on networks that use TCP as a transport, while CoAP is the only protocol that uses UDP, making it the most lightweight and not requiring a handshake method.

While HTTP communication patterns are implemented based on a request-response model, protocols such as MQTT, AMQP, and XMPP are structured using a publish-subscribe communication model, and CoAP implements both. MQTT contains a smaller header of only 2 bytes compared to 4 bytes for CoAP and AMQP, which has a header size of 8 bytes. HTTP headers include various metadata about content type, cookies, requests, or responses, and more.

The size of the HTTP header varies depending on the type of header and the amount of information included in the request or response. Despite its overhead, HTTP continues to be used in IoT devices due to its widespread compatibility with existing systems and infrastructure.

HTTP may transfer a large amount of data in tiny packets, which causes large overhead. Use of HTTP in complex systems can lead to high latency due to the increased number of requests sent periodically. XMPP uses XML messages. XML messages cause additional overhead due to several headers and tags, named traffic overhead. XML overhead increases the power consumption of IoT devices, which is the biggest disadvantage of IoT. MQTT overhead is low when compared to XMPP and HTTP. Due to its low overhead, MQTT is suited for devices with limited resources, making it ideal for scenarios where minimizing data transmission duration is needed. For Payload Format, MQTT, AMQP, and CoAP are binary protocols; HTTP is a text-based protocol.

When the QoS is compared, MQTT implements three qualities of service (QoS) levels, and AMQP also supports three QoS levels. while CoAP limits QoS to two message types: confirmable and non-confirmable. For protocols such as XMPP and HTTP, QoS is not handled at the application layer but is instead provided by the underlying transport protocols, such as TCP.

Although HTTP doesn't offer any further choices, it depends on TCP, and this ensures successful delivery as long as the devices are connected. Since XMPP does not provide QoS features, all messages will be handled similarly. XMPP does not support any delivery guarantees. The performance of the MQTT protocol strongly depends on the quality of service (QoS) profile. It has better QoS as compared to the protocols CoAP and HTTP. The protocol provides strong QoS support, and no other protocol can match QoS 2, which offers the highest level of service in MQTT, and QoS2 ensures that each message is delivered exactly once to the intended recipients. To achieve this, QoS 2 involves a four-part handshake between the sender and receiver (HiveMQ, 2023). The QoS provided by the CoAP is similar to that of QoS 0 of the MQTT (AL-MASRI et al., 2020).

For security issues, almost all of these protocols implement the TLS/SSL or DTLS protocols as their security methods. For CoAP, it uses DTLS on top of its UDP transport protocol. Secure Sockets Layer (SSL) certificates, sometimes called digital certificates, is standard technology for securing an internet connection by encrypting data sent between a website and a browser (or between two servers).

XMPP, CoAP, and HTTP are standardized by the IETF Constrained RESTful Environments (CoRe) Working Group. MQTT and AMQP protocols were developed as open OASIS standards and ISO recommended protocols.

3.4 Conclusion of This Chapter

Through this chapter, the research methodology was applied, where a set of IoT protocols operating in the application layer was presented, and these protocols were compared by introducing their characteristics to determine their best-fit scenarios when applied in the smart city.

The philosophy of HTTP revolves around the transfer of files and documents. It is a text-based protocol designed for communication between only two systems at a time. Although some developers use HTTP to transfer binary data by bypassing its typical functions, this protocol was not originally intended for this purpose. HTTP is an excellent protocol within the Internet space, but its design philosophy focuses on transferring HTML files. HTTP was developed to support request-response communication rather than event-driven communication. However, most IoT applications are event-based.

So, it cannot meet the needs of devices with limited resources and power, such as sensor nodes. In IoT systems, Bytes are transferred, and the protocol that innovates in this field is MQTT. HTTP has several limitations that apply to IOT applications, and many advanced application-layer protocols such as MQTT, AMQP, and CoAP have been created to address these limitations. Therefore, it will be excluded from the application work in this thesis.

CoAP is an excellent protocol for its application in the Internet of Things, but it faces a challenge in that the infrastructure to build a system for the protocol to work must be IPv6. Here, the system developer needs to design and develop a Gateway for data transfer. For this reason, I excluded the protocol from the applications and practical experiments intended for the fourth unit in this study.

Based on the philosophy of the XMPP protocol, it was excluded because it is an XML-based format that requires a large number of bytes, making it not ideal for IoT applications in the smart city due to its verbosity and large size. The philosophy of AMQP (Advanced Message Queuing Protocol) is to provide a strong, flexible, and dependable messaging system that supports complicated message patterns and guarantees. AMQP was originally conceived not within industrial automation circles, but financial sector. This protocol is good for exchanging financial data between banks and the financial services industry.

Chapter 4: Implementation and Experiments

This chapter outlines the implementation process and describes the experiments designed to evaluate the performance of the protocol chosen for this research. To carry out this performance evaluation and analyze the behavior of the protocol, emulate and model the smart city using scenarios similar to those in the real smart city system that contains a continuous data flow application that is similar in its working mechanism to some IoT applications, such as the IoT smart environmental monitoring system, healthcare applications, or smart transportation systems, that collect real-time traffic information through sensors as an example.

4.1 Chosen protocol

In the context of designing IoT system solutions for smart city purposes, after the author made these comparisons between protocols based on the philosophy of each protocol and the original purpose of its design and creation, it is important to answer the following question: Why did the researcher decide to use the MQTT protocol in conducting the experiments?

Unit Two reviews the relevant literature, examines various protocols through theoretical comparisons, and explores the underlying philosophy of each protocol separately. This study concluded that the MQTT protocol is a compromise between HTTP, CoAP, XMPP, and AMQP protocols.

MQTT is a lightweight uses a publish-subscribe model with minimal overhead that includes some important communication features, such as the message delivery assurance mechanisms which allow researchers to evaluate reliability under different network conditions, this is useful for measuring latency and message loss, as done in the experiments, also there are already available libraries that can be reuse to implement the clients and the MQTT broker, other reasons that encourage its use are supported by many platforms, brokers such as HiveMQ, and libraries like Paho-MQTT.

This makes it a practical choice for real-world deployments and experimentation. Real-time data communication is a crucial issue in smart cities, and the MQTT protocol supports this feature for various use cases.

MQTT might be the best option in some circumstances. MQTT is appropriate for low-bandwidth networks. In agreement with past studies, hundreds of IoT applications utilize several types of data transmission protocols, and most of these publications found that MQTT stands out as the most suitable communication protocol for the IoT domain.

"The MQTT protocol is concluded to be the most suitable and cost-effective option for implementing IoT systems due to its strong support through a wide range of middleware, software, and cloud servers based on the outcomes of numerous projects, applications, specifications, and system requirements. The next unit will address the question: How does the selected protocol perform in terms of latency and packet error? What are the reasons that make it particularly suitable for smart city applications?

4.2 Implementation of Smart City Scenario-Based MQTT Data Protocol

4.2.1 System Architecture Overview

Once the protocol has been selected, an environment should be prepared to experiment. A system composed of one Raspberry Pi 4 Computer Model B has been set up. The system can measure, monitor, and report the traffic of data transfer in real time. Raspberry Pi 4 Computer Model B is a popular, effective, low-cost minicomputer. It is suitable for prototyping machine-to-machine solutions. The Raspberry Pi 4 includes a high-performance 64-bit quad-core processor, dual-display support at resolutions up to 4K via a pair of micro-HDMI ports, 8GB of RAM, dual-band 2.4/5.0 GHz wireless LAN, Bluetooth 5.0, Gigabit Ethernet, and USB 3.0. Raspberry Pi requires an operating system to perform; several operating systems are available for the Raspberry Pi. Raspberry Pi OS (previously called Raspbian) is the default Linux distribution for the Raspberry Pi. The 64-bit version of the Raspberry Pi OS with a desktop was opted for; this operating system is based on Debian version 12. To install this operating system, a microSD card is needed. Then, using the "Raspberry Pi Imager" in order to download the OS to the microSD card. Figure 4.1 shows the Raspberry Pi 4 Computer Model B.

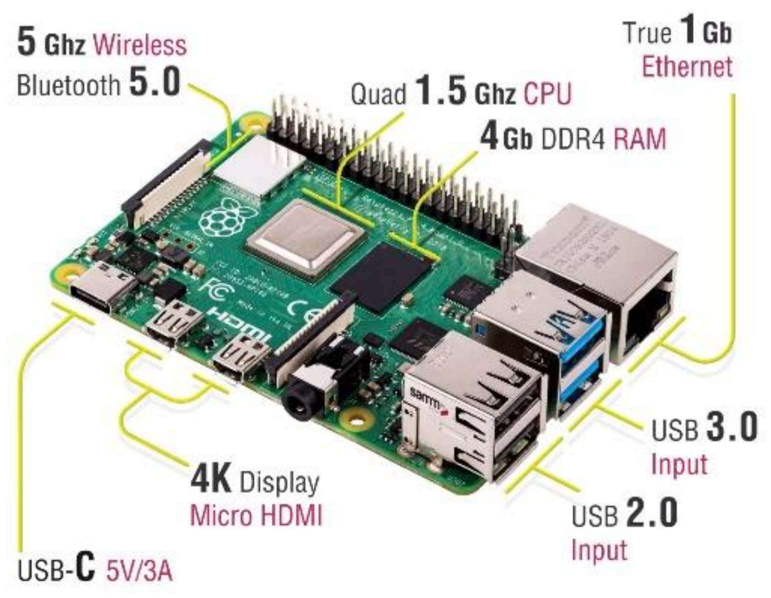


Figure 4.1: Raspberry Pi 4 Computer Model B.

The system architecture shown in Figure 4.2 is the platform layout for all experiments, and it is composed of three layers or parts: two clients and a message broker.

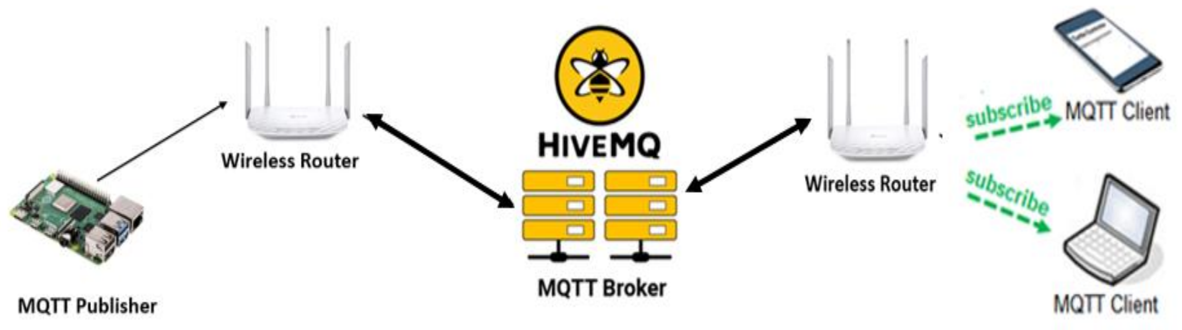


Figure 4.2: The system architecture of IoT

4.2.2 MQTT Publisher

The first part represents the Sensing Layer it is a fixed IoT node device which is simulated by the raspberry pi 4 acts as the publisher located on a remote site sends data packets to a cloud-based broker HiveMQ which it is a public MQTT server (broker) act as a middle point that facilitates communication between the publisher (Raspberry Pi) and the subscriber (laptop) at specified intervals and based on the specified QoS levels, the third part is an endpoint consists of subscriber software to collect the data. A popular MQTT client library for Python, Paho- MQTT, will be used on the publisher side. (Craggs, n.d.-b). Typically, the publisher is a microcontroller that collects measurement data, such as air temperature or other parameters of a system. It publishes the data periodically or is triggered by a specific event, and the information can be retrieved by any number of subscribers.

4.2.3 Broker (Public MQTT Broker)

The MQTT broker is the central part of the entire MQTT protocol architecture. (Rennoch et al., 2020), it acts as the server that passes the messages between the clients. The publisher transmits messages to the broker, which then receives the data and forwards it to the subscriber. Messages are obtained by the subscriber from the broker on a specific topic. MQTT brokers are usually installed in cloud environments where controllers and IoT devices communicate. However, sometimes brokers are installed locally on the devices.

Many MQTT brokers are available, including desktop and cloud-based brokers (Jaloudi, 2019). There is a large variety of MQTT brokers available. Mosquitto is the most widely used broker. In addition to Mosquitto, a few more options are available for our messaging needs. Some popular MQTT brokers include HiveMQ (HiveMQ, 2023), RabbitMQ, EMQX, and AWS IoT Core. See Table 4.1. The online version of the Hive MQ broker will be used because it is free for testing, and installation or configuration is not required. An account should be created, a cluster set up and initialized, and the data posted to the MQTT broker “broker.hivemq.com”. **Public MQTT Broker** and Cloud-based MQTT broker are considered a secure and scalable environment.

Table 4.1: A List of Popular MQTT Brokers

MQTT Broker	Developer	Web site	definition
HIVEMQ	Hive MQ Germany	https://www.hivemq.com/public-mqtt-broker/	HiveMQ is a MQTT broker for M2M and IoT that produces reliable and performant open-source MQTT clients.
Open Automation Software	(OAS) USA	https://openautomationsoftware.com	The OAS platform serves as an IoT Gateway and protocol bus, facilitating the transfer of data between devices, databases, applications, and IoT services.
EMQX	EMQ China	https://www.emqx.com/en/mqtt/public-mqtt5-broker	EMQX is an MQTT broker that is scalable and open-source, with high performance, enabling the connection of over 100 million IoT devices in a single cluster.
MOSQUITTO	Eclipse UK	https://test.mosquitto.org/	An open-source message broker.

4.2.4 MQTT Subscriber

The remote laptop, which acts as another MQTT client ready to receive data packets from the broker in real-time and display them on the monitor, is configured as a subscriber for these topics. The client device used in this study was an HP ProBook 450 G6 laptop running Windows 10 Pro 64-bit. Intel Core i7-8565U CPU @ 1.80 GHz, 8 GB of RAM, a 1024 GB SSD, and an Intel Wireless 7260 Network Adapter. The laptop is connected to the router over Wi-Fi and receives its IP address from the router. The Wireshark program analyzer was used to monitor and capture the MQTT packets transmitted between the devices and the broker.

4.2.5 Python and IoT

Python is a programming language used in IoT for its ease of use and its support of several libraries. It's ideal for industrial control, environmental monitoring, and smart home automation due to its ability to deal with large amounts of data. Python is compatible with microcontrollers, making it a useful tool for developing IoT systems. (Tao, 2024).

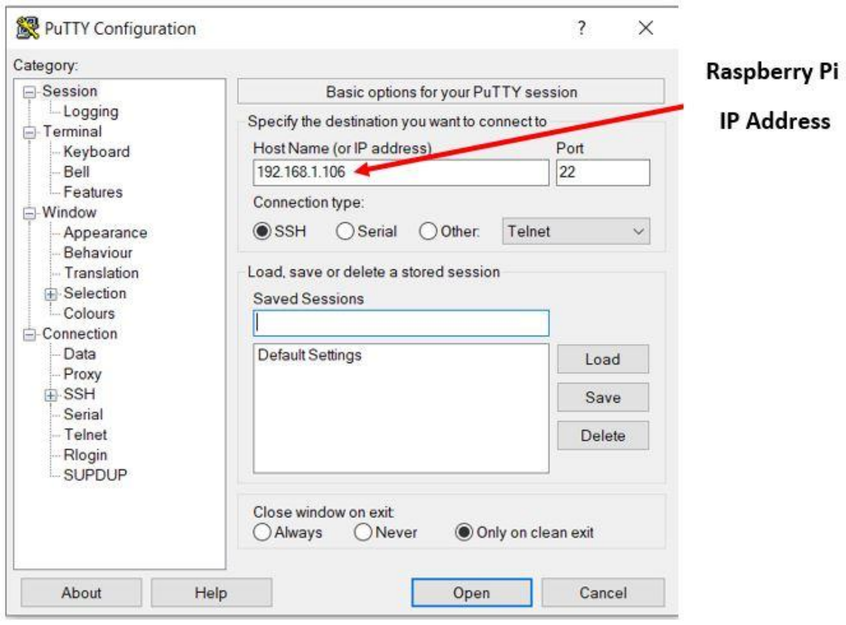
Python is used to implement MQTT, with the Paho-MQTT library as the primary programming library. The Python Paho-MQTT library is a lightweight and ideal solution for Raspberry Pi applications and can be used to transfer data from a Raspberry Pi 4 to an MQTT broker. Paho-MQTT is the most widely used MQTT client library within the Python community; it is an open-source library that offers a simple API and supports various security mechanisms. It is ensured that the Paho-MQTT library is installed on the Raspberry Pi; it should be installed using pip: pip install Paho-MQTT.

A Python script is created to send a payload to the HiveMQ broker. HiveMQ MQTT Client and Paho-MQTT were selected for the chosen protocol, which is MQTT. Therefore, the selected protocol is MQTT, due to its advantages—previously discussed in Sections 3.2.2 and 3.3—such as its simplicity, ease of implementation, and adaptability to high-latency networks. Since MQTT is a publish-subscribe protocol, subscribers must be familiar with the topic names in order to subscribe.

In Python, strings are typically encoded using UTF-8. UTF-8 encoding uses 1 byte for ASCII characters. Thus, the total number of bytes is simply the number of characters in the string. For example, the length of the string "smart" consists of 5 bytes, and the length of the string "smart city" consists of 10 bytes, so each character will use 1 byte.

4.2.6 Network configuration

In this project, as shown before, the publisher is simulated by the Raspberry Pi 4 with a Raspbian operating system, which is the default Linux distribution. The Internet connection of the network is established wirelessly after logging in to the Raspberry Pi via SSH by using an SSH client PuTTY Which is a program used to access the Raspberry Pi command-line interface it uses SSH (secure shell) to open a terminal window on the computer, see figure 4.4, also the laptop is connected to the internet through the same Wi-Fi network. When connected, the Raspberry Pi and the laptop receive a network IP address to enable them to be identified within the internal network among different devices using the same router. To wirelessly connect Raspberry Pi to the router, some changes need to be applied to Raspberry Pi configuration, then the LAN IP is looked up on the Raspberry using ifconfig, the router's IP address is 192.168.1.1, and Raspberry Pi obtained the IP address 192.168.1.106. To ensure the connection has been established between the Raspberry Pi and the router, the laptop should successfully ping the Raspberry Pi IP address using the terminal, with the WiFi bandwidth fixed on the 2.4 GHz band for the router. Figure 4.4 illustrates the message exchange process in MQTT, and Figure 4.3 outlines the steps involved in establishing the communication process in MQTT.



Specify the destination you want to connect to
Host Name (or IP address): 192.168.1.106
Port: 22
Connection type: [✓ SSH] Serial Other: Telnet
Load, save or delete a stored session
Saved Sessions
Default Settings
Close window on exit: Always Never [✓ Only on clean exit]

Figure 4.3: Use PuTTY to Access the Raspberry Pi Terminal from a Computer

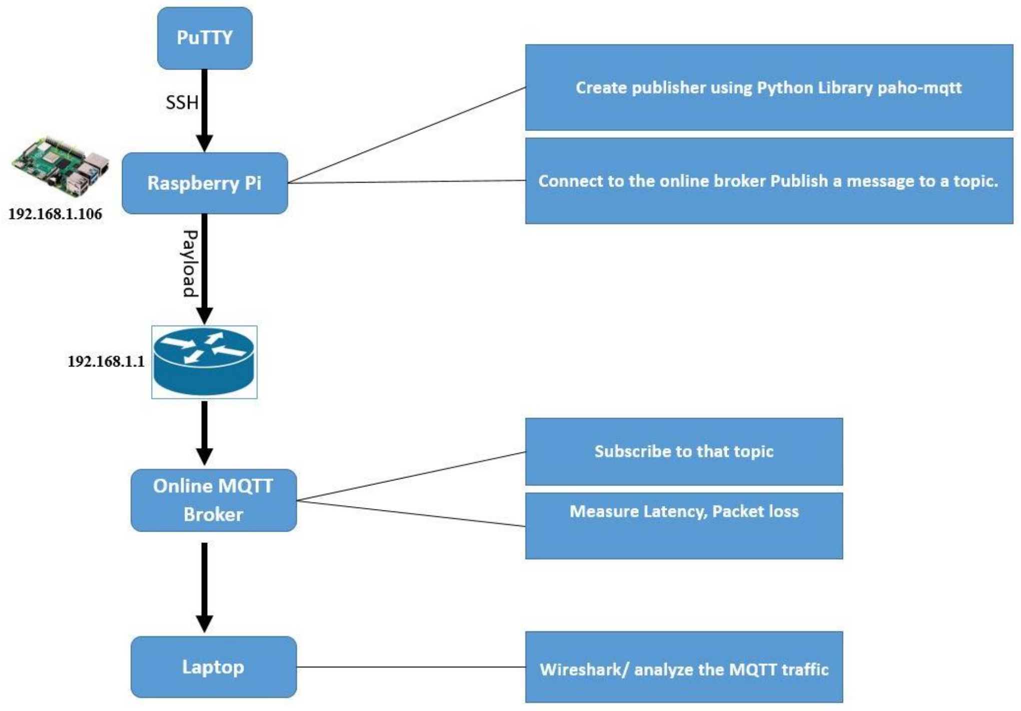

PuTTY
SSH
Raspberry Pi
192.168.1.106
Payload
192.168.1.1
Online MQTT Broker
Laptop
Create publisher using Python Library paho-mqtt
Connect to the online broker Publish a message to a topic.
Subscribe to that topic
Measure Latency, Packet loss
Wireshark/ analyze the MQTT traffic

Figure 4.4: Graph representing the Message exchange process in MQTT

4.2.7 Performance evaluation of IoT MQTT protocol

The goal is to find the performance of MQTT as an IoT protocol. The motivation of these experiments is to identify the issue in the latency effects of the MQTT protocol in the IoT based on network conditions (Analyze Latency). The network latency obtained from the experiment serves as a parameter for evaluating the quality of the network system

Latency is the total time that passes between reading the sensor and the MQTT broker that is receiving the data. In more detail about our experiment, Latency is the period taken from when the publisher sends the data to the online HiveMQ MQTT broker. There are various methods to measure latency, including round-trip and one-way. One-way latency is measured by synchronizing the clocks on both the sender and the receiver and then subtracting the packet transmission time from the packet arrival time at its destination. The one-way approach of latency measurement was used in our test analysis. To measure latency, a timestamp must be embedded in the published message and compared with the timestamp recorded by the broker upon arrival. The payload is published with a timestamp to help in latency calculations on the broker side. The latency is then calculated as the difference between the timestamp recorded by the Raspberry Pi before sending and the arrival time recorded by HiveMQ.

4.2.8 Analyze Packet Sizes

To analyze the obtained results, Wireshark software was utilized for packet capture and analysis to study the behavior of each protocol. Wireshark is the world's foremost network and packet protocol analyzer. The software shows what's happening on the network at a microscopic level (Wireshark, 2023). Wireshark can troubleshoot network issues, capture packets, and monitor network traffic through all types of media in real time. It can analyze a large number of distinct protocols.

Valuable information about the message format and the efficiency of the protocols can be obtained. The packet contents and headers can be visualized using Wireshark. In IoT, Wireshark provides crucial details and insights into the communication between IoT devices and their supporting infrastructure. The main factors influencing the protocols' metrics, such as latency and bandwidth usage, could be identified. Wireshark offers excellent support for MQTT. The values of each MQTT message field are listed (EMQX, 2023). Data for analyzing different graphs are handled using Excel and Python's matplotlib; these curves are explained in each scenario. As a result, the data were collected, and a performance analysis was performed by comparing these results on a graph.

4.2.9 Experiments

Experiment 1: Small Payload Data ranging from 4 bytes to 512 bytes.

The goal of this experiment is to examine and evaluate the performance of the MQTT protocol under different scenarios within smart city applications. To measure latency and message loss, the impact of small payloads on packet transmission times using the MQTT protocol is assessed. In this scenario, various levels of MQTT reliability are tested. The first test iteration involved sending small messages while varying message size by creating strings of different fixed lengths. The payloads ranged from 4 bytes to 512 bytes, specifically 4, 8, 16, 32, 64, 128, 256, and 512 bytes, all with the quality-of-service level QoS 0. The second test iteration was conducted using QoS 1, followed by QoS 2 in the third iteration.

Table 4.2: Packets in Bytes with three levels of QoS for Small Payload.

Iteration Number / Payload (Byte)	1	2	3	4	5	6	7	8
First test (QoS 0)	4	8	16	32	64	128	256	512
Second test (QoS 1)	4	8	16	32	64	128	256	512
Third test (QoS 2)	4	8	16	32	64	128	256	512

Experiment 2: Large Payload Data ranging from 4K bytes to 512K bytes.

This experiment aims to evaluate the performance of the MQTT protocol under different scenarios in the context of smart city applications. The study evaluates latency and message loss by investigating the effects of large payload sizes on the packets' transmission time using the MQTT protocol. In this scenario, the different levels of reliability of MQTT are employed. The first test iteration involved sending large messages by varying the message size through the creation of strings with different fixed lengths. The payloads vary from 4K bytes to 512K bytes, considering 4Kbyte, 8Kbyte, 16Kbyte, 32Kbyte, 64Kbyte, 128Kbyte, 256Kbyte, and 512Kbyte sizes, with the quality-of-service level QoS 0. The second test iteration was conducted using QoS level 1, followed by QoS level 2, and repeated in the third iteration.

Table 4.3: Packets in Bytes with three levels of QoS for large Payload.

Iteration Number / Payload (Byte)	1	2	3	4	5	6	7	8
First test (QoS 0)	4K	8K	16K	32K	64K	128K	256K	512K
Second test (QoS 1)	4K	8K	16K	32K	64K	128K	256K	512K
Third test (QoS 2)	4K	8K	16K	32K	64K	128K	256K	512K

A summary of the experiment setup is presented in Table 4.4.

Table 4.4: Features of test platform.

Publisher	Raspberry Pi 4 (as client).
Broker	HiveMQ (a cloud-based or self-hosted MQTT broker).
Subscriber	Laptop computer.
Protocol	MQTT protocol.
MQTT library	Paho MQTT for Python
Message Sizes	Experiment 1: Small Payload Data ranging from 4 bytes to 512 bytes. 4bytes, 8byte, 16byte, 32byte, 64byte, 128byte, 256byte, 512byte. Experiment 2: Large Payload Data ranging from 4K bytes to 512K bytes. 4Kbyte, 8Kbyte, 16Kbyte, 32Kbyte, 64Kbyte, 128Kbyte, 256Kbyte, 512Kbyte.
QoS Levels	QoS 0, QoS 1, and QoS 2.
Metrics	Latency: The time taken for a message to be sent from the publisher to the subscriber via the HiveMQ broker.
Packet Capture tool	Wireshark

Experiment 3: Broker Software for Payload Data ranging from 4 bytes to 512 bytes.

The researchers recommended several MQTT brokers. Selecting the appropriate broker is another critical factor when implementing an MQTT system. In this experiment, the goal is to learn more about the strengths and weaknesses of a message broker and to evaluate the performance of two open-source brokers (Mosquitto and HiveMQ), which are the most popular among developers at QoS level 0, by examining the latency using the MQTT protocol and studying the effects of payload on a message's transmission time. For better MQTT results, the measured latency is performed for QoS Level 0.

Table 4.5: Features of test platform.

Publisher	Raspberry Pi 4 (as client).
Broker	Mosquitto. HiveMQ.
Subscriber	Laptop computer.
Protocol	MQTT protocol.
MQTT library	Paho MQTT for Python
Message Sizes	Small (4 bytes to 512 bytes.)
QoS Levels	QoS Level: 0 (at-most-once delivery, no ACK).
Metrics	Latency
Packet Capture tool	Wireshark

Experiment 4: Packet loss measurements for small Payload Data ranging from 4 bytes to 512 bytes.

Publisher	Raspberry Pi 4 (as client).
Broker	HiveMQ.
Subscriber	Laptop computer.
Protocol	MQTT protocol.
MQTT library	Paho MQTT for Python
Message Sizes	Small (4 bytes to 512 bytes.)
QoS Levels	QoS Level: 0, 1, 2
Metrics	Packet Loss
Packet Capture tool	Wireshark
# of messages	1000

In this experiment, one thousand messages were sent for each payload size, 4 bytes, 8 bytes, and so on, to analyze message loss in a wireless network environment. The test was repeated using QoS levels 0, 1, and 2."

Chapter 5: Results and Discussions

5.1 Introduction

Chapter 4 presents the experimental test platform developed to evaluate the performance of the MQTT protocol. This platform enabled systematic testing under various conditions, and the results obtained from the experiments are discussed and analyzed in detail within this chapter.

The message payload and QoS will both be used as criteria to test latency. The experiments were conducted according to the scenarios presented in Chapter 4, and then a comparison of the performance of each case was conducted. The impact of varying the payload size and reliability levels on the performance of the MQTT protocol was studied. The performance measurement used to evaluate the scenarios is latency, defined as the time required to publish data from the Raspberry Pi to the broker.

5.2 Results and Findings

The following result is based on using the Raspberry Pi 4 with onboard Wi-Fi to send data, acting as the publisher, while online HiveMQ was the broker, and the protocol used was MQTT. Latency was determined by the time taken for the data to reach the MQTT broker from the source device. Finally, the data was analyzed to study the results. The experiment was implemented, and the performance of the protocol was evaluated for all the QoS levels and various data sizes.

Several network scenarios were defined based on different QoS levels, through which the best-case scenario with extremely minimal latency was identified to be achieved. The theoretical maximum length of an MQTT packet of 268, 435 ,456 bytes, equivalent to 256 MB, was taken into account. The actual topic string has a maximum length of 65536 bytes. This is a limitation set by the MQTT specifications. The timestamps included in the published messages and deducted from the subscriber side timestamps. The Raspberry Pi recorded a timestamp before sending, and HiveMQ recorded the arrival time; then, we calculated the difference between the two readings as a latency.

Latency Calculation: Latency = Broker Receive Time – Raspberry Pi Send Time

Tables 5.1 and 5.2 show the results taken to evaluate the latency between the publisher and broker by increasing the payload across different QoS levels, Figure 5.1. Figure 5.2 represents the diagram results; it is a graph that compares the times influenced by payloads and QoS levels in a wireless network environment.

5.2.1 Latency findings for MQTT QoS 0 / QoS 1 / QoS 2

Major results for Experiment 1: Small Payload Data ranging from 4 bytes to 512 bytes.

Table 5.1: MQTT Latency (ms)for small payload

MQTT Latency (ms)								
	Small Payload (Bytes)							
QoS	4	8	16	32	64	128	256	512
0	42.5	49.3	60.5	62	69	71	73.5	74
1	159	150	181	189	212	280	313	412
2	281	289	298	317	340	421	456	439

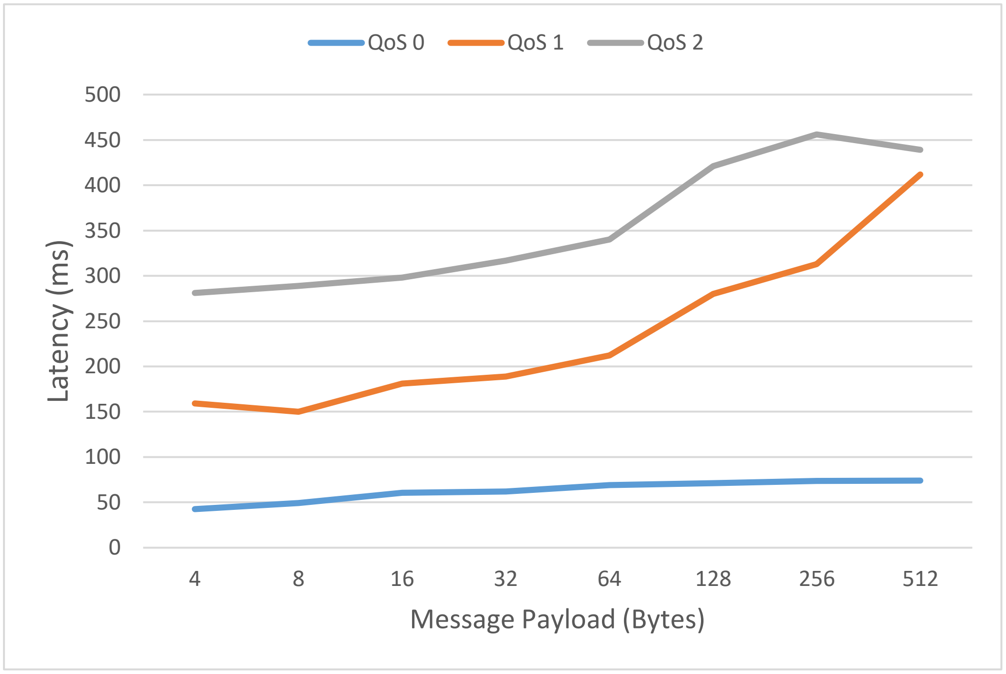


Figure 5.1: Plot latency vs. payload size with three levels of QoS. (4 Byte – 512 Byte)

Major results for Experiment 2: Large Payload Data ranging from 4K bytes to 512K bytes.

Table 5.2: MQTT Latency (ms)for large payload

MQTT Latency (ms)								
	Large Payload (Bytes)							
QoS	4K	8K	16K	32K	64K	128K	256K	512K
0	161	260	281	382	471	485	486	500
1	609	651.5	659	675	678	691.9	698.2	730
2	620	660	673	682	687	698	699	780

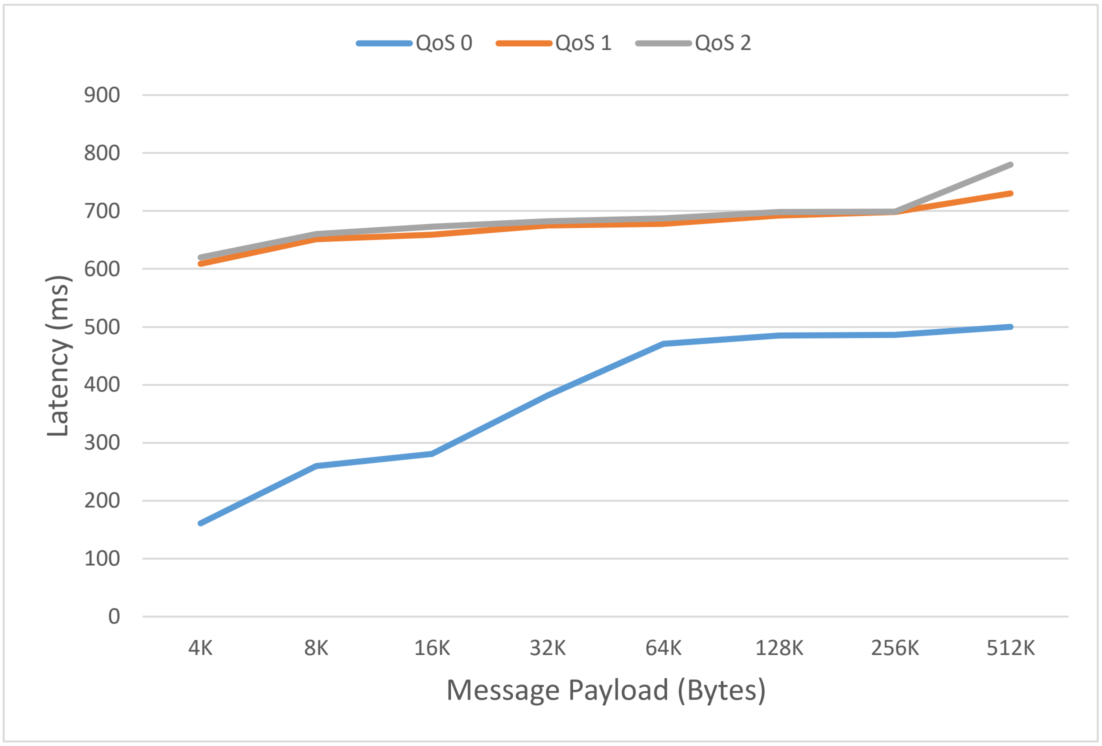


Figure 5.2: Plot latency vs. payload size for each QoS level. (4 KByte – 512 Kbyte)

In Figure 5.1, latency is measured for MQTT with QoS 0, QoS 1, and QoS 2 concerning changing 4byte, 8byte, 16byte, 32byte, 64byte, 128byte, 256byte, and 512byte sizes. In Figure 5.2, latency is measured for MQTT with QoS 0, QoS 1, and QoS 2 concerning changing 4-Kbyte, 8-Kbyte, 16-Kbyte, 32-Kbyte, 64-Kbyte, 128-Kbyte, 256-Kbyte, and 512-Kbyte sizes.

Figures 5.1 and 5.2 show the message latency results for the MQTT protocol across different payload sizes and Quality of Service (QoS) levels. The data shows the relationship between payload size, latency, and QoS level. When payload size increases, message latency also increases, but the extent of increase varies across different QoS levels. QoS 0 has the lowest latency, as messages are transmitted without acknowledgment, resulting in minimal transmission overhead. QoS 1 introduces a slight increase in latency due to its at-least-once delivery mechanism, which requires acknowledgment from the subscriber. QoS 2 shows the highest latency among all levels, as it employs an exactly-once delivery process involving multiple handshake messages to guarantee message delivery without duplication. These findings highlight the trade-off between reliability and performance in MQTT communication, where higher QoS levels ensure greater reliability at the cost of increased latency.

Major results for Experiment 3: The performance of two brokers (Mosquitto and HiveMQ)

Table 5.3: MQTT Latency (ms) of cloud-based broker HiveMQ and Mosquitto

	Payload size							
Payload (Bytes)	4Byte	8Byte	16Byte	32Byte	64Byte	128Byte	256Byte	512Byte
HiveMQ	42.5	49.3	60.5	62	69	71	73.5	74
Mosquitto	50	55.2	66.5	68	75	77	78	81

The results are shown in table 5.3, as can be observed in Figure 5.3, HiveMQ often has lower latency compared to Mosquitto broker.

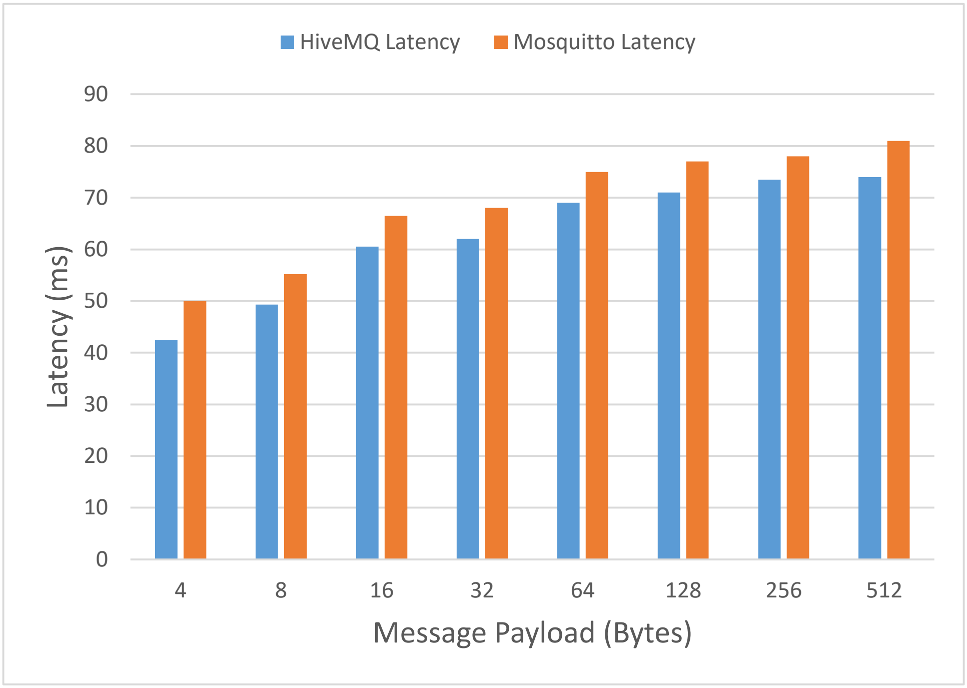


Figure 5.3: MQTT Latency over cloud-based broker HiveMQ and Mosquitto

Major results for Experiment 4: Packet loss measurements for small Payload Data ranging from 4 bytes to 512 bytes.

Packet loss is defined as the number of data packets that fail to reach their destination relative to the total number of packets sent. It is typically expressed as a percentage. In our experiment, Packet loss was calculated by analyzing the MQTT traffic captured with Wireshark. The number of missing or unacknowledged messages was treated as lost packets. This method allowed for accurate packet loss detection across various payload sizes and QoS levels.

$$\text{Packet Loss (\%)} = \left(\frac{\text{Packets Sent} - \text{Packets Received}}{\text{Packets Sent}}\right) \times 100$$

Table 5.4 shows the results of MQTT packet loss regarding QoS 0; Table 5.5 shows the results of MQTT packet loss regarding QoS 1; Table 5.6 shows the results of MQTT packet loss regarding QoS 2.

Table 5.4: Results of MQTT packet loss vs payload for QoS 0

QoS 0								
Payload	4B	8B	16B	32B	64B	128B	256B	512B
sent	1000	1000	1000	1000	1000	1000	1000	1000
received	920	940	935	930	928	933	941	945
Packet Loss	8%	6%	6.5%	7%	7.2%	6.7%	5.9%	5.5%

Table 5.5: Results of MQTT packet loss vs payload for QoS 1

QoS 1								
Payload	4B	8B	16B	32B	64B	128B	256B	512B
sent	1000	1000	1000	1000	1000	1000	1000	1000
received	995	996	995	997	992	995	993	995
Packet Loss	0.5 %	0.4%	0.5%	0.3%	0.8%	0.5%	0.7%	0.5%

Table 5.6: Results of MQTT packet loss vs payload for QoS 2

QoS 2								
Payload	4B	8B	16B	32B	64B	128B	256B	512B
sent	1000	1000	1000	1000	1000	1000	1000	1000
received	999	998	998	998	999	997	996	999
Packet Loss	0.1%	0.2%	0.2%	0.2%	0.1%	0.3%	0.4%	0.1%

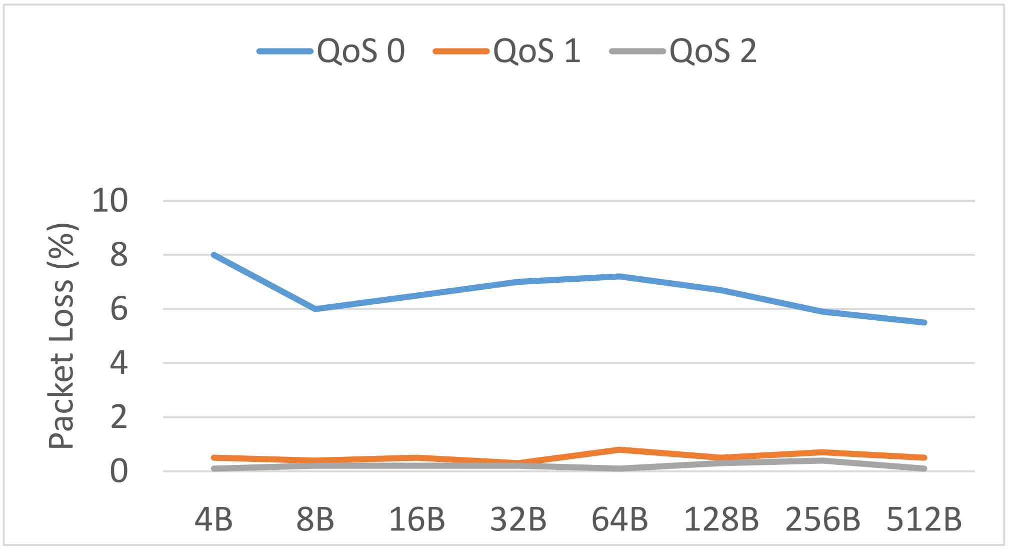


Figure 5.4: MQTT Packet loss measurements with three QoS analysis results

5.3 Discussion about the results

A discussion of the results from section 5.2 was presented in this section. To summarize, the MQTT packet delivery was evaluated using a real wireless platform, which was composed of three layers: two clients and one message broker (a subscribe client, a broker, and a publish client). MQTT supports three QoS levels that ensure the delivery of messages reliably. In order to conduct this performance evaluation and study the behavior of the MQTT protocol, the latency and packet loss were examined, and packets were recorded while messages were sent with different payload sizes across three different QoS levels.

The first scenario is to employ the MQTT protocol with a small payload ranging from 4 bytes to 512 bytes, varying the quality-of-service QoS 0, QoS 1, and QoS 2. This example could be a simulation of a traffic monitoring system, which does not require a large payload of data. Examples of the amount of payload are vehicle count and timestamp, which are small payloads (20–100 bytes).

In the second scenario, large MQTT payload data ranging from 4K bytes to 512K bytes are sent using varying QoS 0, QoS 1, and QoS 2. This example could be a simulation of a smart metering system, which requires a large payload of data. Examples of the amount of payload are encrypted usage data logs, which are large payloads (4KB to 1MB). Of course, if the data is collected for a certain period, for example, an hour, then it is sent in a batch file

The metric analyzed is the latency, the latency measurements showed that for the tests carried out on the MQTT Protocol, as can be seen in Figures 5.1 and 5.2, each figure presents the message latency results for various payloads of the MQTT protocol, it shows how payload size, latency, and QoS level relate to each other, for example the latency for a 64 Bytes message is lower than the latency for a message of 128 Bytes for all types of QoS. QoS 0 has the shortest transmission time, followed by QoS 1, then QoS 2. QoS 0 provides minimal latency per packet because it only sends once, and QoS 1 has a slightly higher latency since it delivers at least once. With its exactly once delivery, QoS 2 has the highest latency because it sends a series of messages and uses a 4-way handshake to ensure the arrival of the message. The latency increases linearly with the payload size from 4 bytes to 512 bytes. It increased from 42.5 ms for 4 bytes to 74 ms for 512 bytes in QoS 0. The latency increased from 159 ms for 4 bytes to 412 ms for 512 bytes in QoS 1. The latency increased from 281 ms for 4 bytes to 439 ms for 512 bytes in QoS 2. Table 5.7 summarizes the effect of payload size vs QoS on latency for MQTT.

Table 5.7: effect of payload size vs QoS ON Latency for MQTT

Payload size	QoS 0	QoS 1	QoS 2
Small (4Byte-512Byte)	Lowest latency	Low latency	Moderate latency
Large (4KByte-512KByte)	Moderate latency	High latency	Very high latency

Higher QoS levels generally increase reliability but also introduce extra latency due to the additional handshakes. The latency appears to increase proportionally with increasing QoS levels and payload size; in other words, there is a direct proportional relationship between the time and payload size across all the QoS levels.

So, it was expected that the lower the number of milliseconds and latency, the more efficient the network would behave. Therefore, the user will have a better experience, and larger payloads will naturally result in higher latency, especially in low-bandwidth environments. As a result, consider testing with varying payload sizes. It was found that the use of QoS 1 is best balanced between reliability and speed, considering the reduction of the payload to achieve the lowest possible latency time.

Based on the related works discussed in section 2.2, it was found that the performance of the MQTT protocol strongly depends on the Quality of Service (QoS) profile, as mentioned in [(Mijovic et al., 2016), (Marques et al., 2018)]. The research results are somewhat consistent with the results of previous studies.

Figure 5.3 Mosquitto shows lower latency in QoS 0 and small payloads due to its lightweight nature; HiveMQ exhibits slightly higher latency. When comparing the performance of Mosquitto and HiveMQ brokers, several factors come into play, such as latency and message loss. HiveMQ is better with large payloads than Mosquitto. These results are consistent with many previous studies, including the study of (Jaloudi, 2019) and the study (Mishra & Reddy, 2024).

Latency is an important factor that is impacted by the message size in many types of IoT wireless systems, and its effect will vary depending on the application scenario. Reducing latency in IoT design improves performance and improves the user experience for these applications. Minimizing latency is one of the primary concerns when designing an Internet of Things (IoT) system.

A protocol's performance can be evaluated using a variety of metrics, each reflecting different aspects of system suitability for specific use cases related to smart city applications. These metrics can be grouped into three main categories: the first category includes Time-Based Metrics include Latency (Delay), which is the time it takes for the packet to travel from its starting device to its destination device, Response Time, Round-Trip Time (RTT), and Jitter. The second category is Packet and Transmission Metrics: This category focuses on the structure of data transmission, including payload size, protocol overhead, packet loss, and throughput. The third category is Energy metrics, including energy consumption: the amount of energy consumed during message transmission and reception. Bandwidth Efficiency: The ratio of useful data to total transmitted data.

Chapter 6: Conclusion

This thesis is a research effort that helped to develop a better understanding of IoT messaging protocols and gain a deeper understanding of implementing these protocols in smart cities. A review of the Internet of Things and how it relates to smart cities, including the definition and fundamentals of IoT and its architecture, was presented in this thesis. The smart city concept was first presented, and important terms of the Internet of Things were discussed. The IoT application layer protocols were presented, recent works related to this thesis topic were discussed, and then a comparative analysis of the application layer protocols was conducted.

Through a review of scientific research papers and a comparative theoretical analysis of various IoT application layer protocols, five protocols were presented and compared conceptually in terms of different performance metrics. These protocols are MQTT, CoAP, XMPP, AMQP, and HTTP. Among the conclusions drawn from this extensive review, it was found that various factors can influence a protocol evaluation; however, it might be challenging to provide an accurate description of performance.

Latency is one of the most important factors to study and evaluate when comparing different communication protocol parameters, especially in the world of IoT applications. As a result, care must be taken to minimize latency between IoT nodes. Latency varies depending on the IoT protocol; however, the general conclusion that can be drawn concerning the applicable scenario is that all IoT application layer protocols are of appropriate latency, and real-time and high-performance applications require a low-latency protocol. Protocol, payload size, and quality of service are all crucial test platform factors to consider while planning and building an IoT system. In summary, the evaluation of protocol performance during the implementation of the IoT system is an important factor that needs to be taken into account.

The results reveal that IoT application layer protocols were specifically built for M2M communication. The challenge of choosing the right IoT protocol for the right application requires comprehensive planning and analysis of the problem domains that need to be solved. For example, the application of IoT in designing traffic monitoring within smart cities completely differs from the application of IoT for medical purposes. No singular IoT protocol solution can be the best choice for all types of IoT systems. Choosing the right real-time protocol depends largely on the needs of a given developer and their IoT devices.

Based on the theoretical analysis and comparison of various IoT communication protocols, this study concludes that MQTT, which is a binary protocol, represents a balanced compromise among the protocols studied during this work. This thesis identifies MQTT as a balanced and practical solution. It focuses on MQTT because it combines the lightweight nature and publish-subscribe paradigm with the low overhead needed for constrained environments. The method of communication is basic; messages can be exchanged at any time, and it ensures message reliability by providing three QoS levels. The publish-subscribe model enables asynchronous, decoupled communication between devices, making it highly suitable for dynamic and large-scale smart city applications.

During the period from 2016 to 2024, which serves as the timeframe for the literature review conducted in this study, MQTT has experienced a significant rise in adoption across the IoT domain. Multiple studies and industry reports confirm that MQTT consistently ranks among the top three most widely used messaging protocols for IoT applications, making it a preferred choice for both researchers and industry practitioners working in smart city and real-time data environments. But like any protocol, MQTT has limitations, and it might not be the best option in every IoT application. Choosing the most appropriate protocol depends on the scenario and requirements.

Finally, this research work can significantly support IoT application developers in making informed decisions while selecting communication protocols. As a result of our study, the MQTT standard is suitable for Internet of Things applications.

6.1 Future work

Future research should focus on understanding how protocols scale among nodes, gateways, brokers, and applications. In this study, the researcher did not address the issue of security and protection, or any predictive analytics technologies within IoT protocols, which are crucial and recommended for future research.

References

Al Mansoori, S. (2021, February). Challenges and New Research Directions to the Development of Smart Cities: Systems-of-Systems Perspective. In Journal of Physics: Conference Series (Vol. 1828, No. 1, p. 012136). IOP Publishing.

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. IEEE communications surveys & tutorials, 17(4), 2347-2376.

Al-Masri, E., Kalyanam, K.R., Batts, J., Kim, J., Singh, S., Vo, T., & Yan, C. (2020). Investigating Messaging Protocols for the Internet of Things (IoT). IEEE Access, 8, 94880-94911.

Alobaidy, Haider & Mandeep, J. & Behjati, Mehran & Nordin, Rosdiadee & Abdullah, N.F.. (2022). Wireless Transmissions, Propagation and Channel Modelling for IoT Technologies: Applications and Challenges. IEEE Access. 10. 1-1. 10.1109/ACCESS.2022.3151967.

Al-Qaseemi, S. A., Almulhim, H. A., Almulhim, M. F., & Chaudhry, S. R. (2016). IoT architecture challenges and issues: Lack of standardization. https://doi.org/10.1109/ftc.2016.7821686

Amazon Web Services, Inc. (2024). *Throughput vs Latency - Difference Between Computer Network Performances.* Retrieved March 5, 2024 from https://aws.amazon.com/compare/the-difference-between-throughput-and-latency/.

Andy, Syaiful & Rahardjo, Budi & Hanindhito, Bagus. (2017). Attack scenarios and security analysis of MQTT communication protocol in IoT system. 1-6. 10.1109/EECSI.2017.8239179.

Bansal, S., & Kumar, D. (2019, July). IoT application layer protocols: performance analysis and significance in smart city. In 2019 10th international conference on computing, communication and networking technologies (ICCCNT) (pp. 1-6). IEEE.

Bansal, S., & Kumar, D. (2020). IoT ecosystem: A survey on devices, gateways, operating systems, middleware and communication. International Journal of Wireless Information Networks, 27(3), 340-364.

Bayılmış, C., Ebleme, M. A., Çavuşoğlu, N., Küçük, K., & Sevin, A. (2022). A survey on communication protocols and performance evaluations for Internet of Things. Digital Communications and Networks, 8(6), 1094–1104. https://doi.org/10.1016/j.dcan.2022.03.013

Becker, Markus. (2013). A cheatsheet for the Constrained Application Protocol (CoAP).

Bellini, P., Nesi, P., & Pantaleo, G. (2022). IoT-enabled smart cities: A review of concepts, frameworks and key technologies. Applied Sciences, 12(3), 1607.

Bluetooth® Technology. (2024). *Bluetooth® Technology Website.* Retrieved January 3, 2024 from *https://www.bluetooth.com/.*

Chandramouli, V. (2002). A Detailed Study on Wireless LAN Technologies.

Cisco, (2014). *"Fast Innovation requires Fast IT," cicso.com.* Retrieved December 30, 2023 from *https://www.cisco.com/c/dam/global/en_ph/assets/ciscoconnect/pdf/bigdata/jim_green_cisco_connect.pdf*.

Cohen, G. S. (2023, July 17). The 6 Key Elements of IoT Cellular Connectivity - FirstPoint. FirstPoint. https://www.firstpoint-mg.com/blog/the-6-key-elements-of-iot-cellular-connectivity/

Cope, B. S. (2022, July 6). *Beginners Guide To The MQTT Protocol.* |. Retrieved December 17, 2023 from *http://www.steves-internet-guide.com/mqtt/.*

Corak, B. H., Okay, F. Y., Guzel, M., Murt, S., & Ozdemir, S. (2018). Comparative analysis of IoT communication protocols. 2022 International Symposium on Networks, Computers and Communications (ISNCC), 1–6. https://doi.org/10.1109/isncc.2018.8530963

Craggs, I. (2024*). MQTT vs HTTP for IoT.* Retrieved March 17, 2024 from *https://www.hivemq.com/article/mqtt-vs-http-protocols-in-iot-iiot.*

Dameri, R. P. (2013). Searching for smart city definition: a comprehensive proposal. International Journal of computers & technology, 11(5), 2544-2551.Dash, Biswajit & Peng, Jun. (2022). Zigbee Wireless Sensor Networks: Performance Study in an Apartment-Based Indoor Environment. Journal of Computer Networks and Communications. 2022. 1-14. 10.1155/2022/2144702. DOI: https://doi.org/10.54808/JSCI.21.04.1

Dragičević, T., Siano, P., & Prabaharan, S. S. (2019). Future generation 5G wireless networks for smart grid: A comprehensive review. Energies, 12(11), 2140.

Ed, M. B. (2022, June 6). *RFC 9114: HTTP/3. IETF Datatracker*. Retrieved March 18, 2024 from *https://datatracker.ietf.org/doc/html/rfc9114.*

Eggly, G. M., Finochietto, M., Dimogerontakis, E., Santos, R. M., Orozco, J., & Meseguer, R. (2018, October). Real-time primitives for coap: Extending the use of iot for time constraint applications for social good. In Proceedings (Vol. 2, No. 19, p. 1257). MDPI.

Embien Technologies::Blog. (2023). *Embien.* Retrieved January 18, 2024 from *https://www.embien.com/blog/introduction-to-lora-technology.*

Ericsson, A. (2016). Cellular networks for massive IoT-enabling low power wide area applications. no. January, 1-13.

Ertürk, M. A., Aydın, M. A., Büyükakkaşlar, M. T., & Evirgen, H. (2019). A survey on LoRaWAN architecture, protocol and technologies. Future internet, 11(10), 216.

Extensible messaging and presence protocol (XMPP). (2023). *XMPP | The universal messaging standard.* Retrieved December 18, 2023 from https://xmpp.org/rfcs/rfc3920.html.

Gerber, A., & Romeo, J. Connecting all the things in the Internet of Things IBM Developer, 2020. URL: https://developer. ibm. com/technologies/iot/articles/iot-lp101-connectivity-network-protocols/(accessed: 30.03. 2021).

Gheorghe-Pop, I. D., Kaiser, A., Rennoch, A., & Hackel, S. (2020, December). A performance benchmarking methodology for MQTT broker implementations. In 2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C) (pp. 506-513). IEEE.

Guth, J., Breitenbücher, U., Falkenthal, M., Leymann, F., & Reinfurt, L. (2016, November). Comparison of IoT platform architectures: A field study based on a reference architecture. In 2016 Cloudification of the Internet of Things (CIoT) (pp. 1-6). IEEE.

Hantrakul, K., Sitti, S., & Tantitharanukul, N. (2017, March). Parking lot guidance software based on MQTT Protocol. In 2017 International Conference on Digital Arts, Media and Technology (ICDAMT) (pp. 75-78). IEEE.

HiveMQ Community Edition (2020). Retrieved December 10, 2023 from https://www.hivemq.com/blog/hivemq-ce-2020-3-released/.

Hofer, J., & Pawaskar, S. (2018, July). Impact of the Application Layer Protocol on Energy Consumption, 4G Utilization and Performance. In 2018 3rd Cloudification of the Internet of Things (CIoT) (pp. 1-7). IEEE.

HTTP response status codes (2023). *MDN Web Docs*. Retrieved December 10, 2023 from https://developer.mozilla.org/en-US/docs/Web/HTTP/Status.

https://datatracker.ietf.org/doc/html/rfc6455.

https://www.raspberrypi.com/documentation/computers/getting-started.html.

ILYAS, M. (2023). Smart Cities: Challenges and Opportunities. Journal of Systemics, Cybernetics and Informatics, 21(4), 1-6.

Interoperability. (2023). *Federal Communications Commission.* Retrieved December 23, 2023 from *https://www.fcc.gov/general/interoperability.*

IoT connected devices worldwide 2019-2030 | Statista. (2023, July 27). *Statista.* Retrieved January 22, 2024 from *https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.*

ITU Internet Reports 2005: The Internet of Things. (2023). *ITU.* Retrieved February 21, 2024 from *https://www.itu.int/pub/S-POL-IR.IT-2005/e.*

Jaloudi, S. (2019). MQTT for IoT-based applications in smart cities. المجلة الفلسطينية للتكنولوجيا والعلوم التطبيقية, (2).

Jaloudi, S. (2019c). A Bridge between Legacy Wireless Communication Systems and Internet of Things. Indonesian Journal of Electrical Engineering and Informatics (IJEEI), 7(2). https://doi.org/10.11591/ijeei.v7i2.979

Kanellopoulos, D., Sharma, V. K., Panagiotakopoulos, T., & Kameas, A. (2023). Networking architectures and protocols for IoT applications in smart cities: Recent developments and perspectives. Electronics, 12(11), 2490.

Kassem, I., & Sleit, A. (2020, June). Elapsed time of IoT application protocol for ECG: a comparative study between CoAP and MQTT. In 2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE) (pp. 1-6). IEEE.

Kayal, P., & Perros, H. (2017, March). A comparison of IoT application layer protocols through a smart parking implementation. In 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN) (pp. 331-336). IEEE.

Khalil, K., Elgazzar, K., & Bayoumi, M. (2018, December). A comparative analysis on resource discovery protocols for the internet of things. In 2018 IEEE Global Communications Conference (GLOBECOM) (pp. 1-7). IEEE.

Kumar, N., & Jamwal, P. (2021). Analysis of modern communication protocols for IoT applications. Karbala International Journal of Modern Science, 7(4), 14.

Lai, C. S., Jia, Y., Dong, Z., Wang, D., Tao, Y., Lai, Q. H., ... & Lai, L. L. (2020). A review of technical standards for smart cities. Clean Technologies, 2(3), 290-310.

Light, R. A. (2017). Mosquitto: server and client implementation of the MQTT protocol. Journal of Open Source Software, 2(13), 265.Margelis, George & Piechocki, R.J. & Kaleshi, Dritan & Thomas, Paul. (2015). Low Throughput Networks for the IoT: Lessons learned from industrial implementations. 181-186. 10.1109/WF-IoT.2015.7389049.

Marques, P., Manfroi, D., Deitos, E., Cegoni, J., Castilhos, R., Rochol, J., ... & Kunst, R. (2019). An IoT-based smart cities infrastructure architecture applied to a waste management scenario. Ad Hoc Networks, 87, 200-208.

Martikkala, A., Lobov, A., Lanz, M., & Ituarte, I. F. (2021). Towards the interoperability of IoT platforms: a case study for data collection and data storage. IFAC-PapersOnLine, 54(1), 1138-1143.

Mehmood, Y., Ahmad, F., Yaqoob, I., Adnane, A., Imran, M., & Guizani, S. (2017). Internet-of-things-based smart cities: Recent advances and challenges. IEEE Communications Magazine, 55(9), 16-24.

Mijovic, S., Shehu, E., & Buratti, C. (2016, September). Comparing application layer protocols for the Internet of Things via experimentation. In 2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI) (pp. 1-5). IEEE.

Mishra, M., & Reddy, S. R. N. (2024). Performance assessment and comparison of lightweight d2d-iot communication protocols over resource constraint environment. Multimedia Tools and Applications, 1-30.

Mosquitto. (2022). *Eclipse Mosquitto* Retrieved February 11, 2024 from https://mosquitto.org/.

MQTT Version 3.1.1. (2023). specification of the Open Document Format. Retrieved February 11, 2024 from https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html.

MQTT Version 5.0. (2023). Retrieved February 11, 2024 from https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

Naik, N. (2017, October). Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In 2017 IEEE international systems engineering symposium (ISSE) (pp. 1-7). IEEE.

Noreen, U., Bounceur, A., & Clavier, L. (2017, May). A study of LoRa low power and wide area network technology. In 2017 International Conference on Advanced Technologies for Signal and Image Processing (ATSIP) (pp. 1-6). IEEE.

Nwankwo, E., David, M., & Onwuka, E. N. (2024). Integration of MQTT-SN and CoAP protocol for enhanced data communications and resource management in WSNs. Bulletin of Electrical Engineering and Informatics, 13(3), 1613-1620.Palmese, Fabio & Longo, Edoardo & Redondi, Alessandro & Cesana, Matteo. (2021). CoAP vs. MQTT-SN: Comparison and Performance Evaluation in Publish-Subscribe Environments. 153-158. 10.1109/WF-IoT51360.2021.9595725.

Paolone, G., Iachetti, D., Paesani, R., Pilotti, F., Marinelli, M., & Di Felice, P. (2022). A holistic overview of the Internet of Things ecosystem. IoT, 3(4), 398-434.

Patti, G., Leonardi, L., Testa, G., & Bello, L. L. (2024). PrioMQTT: A prioritized version of the MQTT protocol. Computer Communications, 220, 43-51.

Qiu, T., Chen, N., Li, K., Atiquzzaman, M., & Zhao, W. (2018). How can heterogeneous internet of things build our future: A survey. IEEE Communications Surveys & Tutorials, 20(3), 2011-2027.

Raj, A., & Shetty, S. D. (2022). IoT eco-system, layered architectures, security and advancing technologies: A comprehensive survey. Wireless Personal Communications, 122(2), 1481-1517.

Raspberry. (2024) *Raspberry Pi Documentation.* Retrieved January 7, 2024 from

RFC 6455: The WebSocket Protocol. (2011). *IETF Datatracker*. Retrieved March 5, 2024 from

RFC 7252: The Constrained Application Protocol (CoAP). (2014, June 26). *IETF Datatracker.* Retrieved March 5, 2024 from https://datatracker.ietf.org/doc/html/rfc7252.

Round Trip Time (RTT) - MDN Web Docs Glossary: Definitions of Web-related terms | MDN. (2023, July 31). *MDN Web Docs.* Retrieved March 5, 2024 from https://developer.mozilla.org/en-US/docs/Glossary/Round_Trip_Time.

Santos, M. G. D., Ameyed, D., Petrillo, F., Jaafar, F., & Cheriet, M. (2020). Internet of Things architectures: A comparative study. arXiv preprint arXiv:2004.12936.

Schiller, E., Aidoo, A., Fuhrer, J., Stahl, J., Ziörjen, M., & Stiller, B. (2022). Landscape of IoT security. Computer Science Review, 44, 100467.

Sikic, Lucija & Janković, Jasna & Afric, Petar & Silic, Marin & Ilic, Zeljko & Pandzic, Hrvoje & Zivic, Marijan & Dzanko, Matija. (2020). A Comparison of Application Layer Communication Protocols in IoT-enabled Smart Grid. 83-86. 10.1109/ELMAR49956.2020.9219030.

Sikimić, M., Amović, M., Vujović, V., Suknović, B., & Manjak, D. (2020, March). An overview of wireless technologies for IoT network. In 2020 19th International Symposium INFOTEH-JAHORINA (INFOTEH) (pp. 1-6). IEEE.

Spohn, M. A. (2022). On MQTT scalability in the Internet of Things: issues, solutions, and future directions. Journal of Electronics and Electrical Engineering, 4-4.

Sultana, T., & Wahid, K. A. (2019). Choice of application layer protocols for next generation video surveillance using internet of video things. IEEE Access, 7, 41607-41624.

Tamizan, Mohd. (2020). Latency Issues in Internet of Things: A Review of Literature and Solution. International Journal of Advanced Trends in Computer Science and Engineering. 9. 83-91. 10.30534/ijatcse/2020/1291.32020.

Tao, D. (2024.). *MQTT in Python with Paho Client: Beginner's Guide 2024. www.emqx.com.* Retrieved February 11, 2024 from https://www.emqx.com/en/blog/how-to-use-mqtt-in-python.

Team, H. (2024, February 20). *What is MQTT Quality of Service (QoS) 0,1, & 2? – MQTT Essentials: Part 6.* Retrieved March 5, 2024 from https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/.

Tran, K. T. M., Pham, A. X., Nguyen, N. P., & Dang, P. T. (2024). Analysis and Performance Comparison of IoT Message Transfer Protocols Applying in Real Photovoltaic System. International Journal of Networked and Distributed Computing, 1-13.

Tsvetanov, F. A., & Pandurski, M. N. (2022). Selection of Protocols for Integration of Sensory Data Networks in Cloud Structures. Int. J. Online Biomed. Eng., 18(9), 29-40.

Tutorialspoint. (2024). *What is Payload in Computer Network?* Retrieved March 5, 2024 from https://www.tutorialspoint.com/what-is-payload-in-computer-network.

Vangelista, L., Zanella, A., & Zorzi, M. (2015). Long-range IoT technologies: The dawn of LoRa™. In Future Access Enablers for Ubiquitous and Intelligent Infrastructures: First International Conference, FABULOUS 2015, Ohrid, Republic of Macedonia, September 23-25, 2015. Revised Selected Papers 1 (pp. 51-58). Springer International Publishing.

Wang, H., Xiong, D., Wang, P., & Liu, Y. (2017). A lightweight XMPP publish/subscribe scheme for resource-constrained IoT devices. IEEE Access, 5, 16393-16405.

Whaiduzzaman, M., Barros, A., Chanda, M., Barman, S., Sultana, T., Rahman, M. S., ... & Fidge, C. (2022). A review of emerging technologies for IoT-based smart cities. Sensors, 22(23), 9271.

Wi-Fi Alliance. (2023). Wi-Fi Alliance Retrieved March 5, 2024 from https://www.wi-fi.org/

Wireshark User's Guide. (2023). *Wireshark User's Guide* Retrieved March 5, 2024 from *https://www.wireshark.org/docs/wsug_html_chunked/index.html*

Wukkadada, B., Wankhede, K., Nambiar, R., & Nair, A. (2018, July). Comparison with HTTP and MQTT in Internet of Things (IoT). In 2018 International Conference on Inventive Research in Computing Applications (ICIRCA) (pp. 249-253). IEEE.

XMPP Standards Foundation. *Extensible Messaging and Presence Protocol. 2021*. Retrieved August 10, 2023 from *https://xmpp.org.*

Yalçınkaya, F., Aydilek, H., Erten, M. Y., & İnanç, N. (2020). IoT based smart home testbed using MQTT communication protocol. International Journal of Engineering Research and Development, 12(1), 317-324.

Yassein, M. B., & Shatnawi, M. Q. (2016, September). Application layer protocols for the Internet of Things: A survey. In 2016 International Conference on Engineering & MIS (ICEMIS) (pp. 1-4). IEEE.

Yousuf, T., Mahmoud, R., Aloul, F., & Zualkernan, I. (2015). Internet of things (IoT) security: current status, challenges and countermeasures. International Journal for Information Security Research (IJISR), 5(4), 608-616.

Zhao, X., Askari, H., & Chen, J. (2021). Nanogenerators for smart cities in the era of 5G and Internet of Things. Joule, 5(6), 1391-1431.

Zhong, X., Yao, R., Chen, C., & Zhu, Y. (2018, July). Research on Scalable Zigbee Wireless Sensor Network Expansion Solution. In IOP Conference Series: Materials Science and Engineering (Vol. 394, No. 3, p. 032071). IOP Publishing.

Zorkany, M., Fahmy, K., & Yahya, A. (2019). Performance evaluation of iot messaging protocol implementation for e-health systems. International Journal of Advanced Computer Science and Applications, 10(11).