Implementation and Evaluation of QUIC protocol on Internet of Things Monitoring Solutions

Versão final após defesa

Luís Filipe Pereira de Almeida

Relatório do Projeto de Dissertação
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor Bruno Miguel Correia da Silva

janeiro de 2025

Declaração de Integridade

Eu, Luís Filipe Pereira de Almeida, que abaixo assino, estudante com o número de inscrição M13006 de/o Mestrado em Engenharia Informática da Faculdade de Engenharias, declaro ter desenvolvido o presente trabalho e elaborado o presente texto em total consonância com o Código de Integridades da Universidade da Beira Interior.

Mais concretamente afirmo não ter incorrido em qualquer das variedades de Fraude Académica, e que aqui declaro conhecer, que em particular atendi à exigida referenciação de frases, extratos, imagens e outras formas de trabalho intelectual, e assumindo assim na íntegra as responsabilidades da autoria.

Universidade da Beira Interior, Covilhã 10/01/2025

Acknowledgments

After all the work put into this report, I would first like to give a special thanks to my family for the support they have provided me throughout this journey and for always providing me with the means to work. I would also like to thank all my friends who supported me, not only in the completion of this report but also throughout my academic journey.
A special thanks to my advisor, Professor Doctor Bruno Miguel Correia da Silva, for his guidance, support, and patience throughout the entire process. His knowledge and dedication were fundamental to the development of this work. Also, this work was parcially supported by the Instituto de Telecomunicações, Covilhã, Department of Informatics of the University of Beira Interior. It was developed in the Network Applications and Services-CV Research Cluster, sins-Lab.
Finally, I would like to thank the University of Beira Interior for everything it has allowed me to experience during these years and to all those who have been involved, in one way or another, in this project.

Abstract

IoT is a technology that ensures the possibility of significantly improving our lives by connecting hardware devices and everyday objects to the internet. Subsequently, these devices will be used to collect, transfer, and analyze vast amount of data, which can then be used to act on our environment or interact with us directly. The market for this technology is rapidly expanding in various sectors such as transportation, manufacturing, healthcare, among others. Due to this, IoT is also undergoing rapid evolution, requiring the development of fast, secure, and reliable communications.
In an attempet to address the issues refered before, the use of QUIC in IoT technology is being explored as an alternative to currently used internet transport protocols. Although initially designed to operate in conjunction with HTTP/3, it has features that can benefit IoT, such as low latency compared to the TCP protocol, multiplexing that resolves head-of-line blocking found in TCP, and the use of a connection ID to enable connection migration.
This dissertation explores the implementation of the QUIC protocol on IoT monitoring solutions. We aim to develop, implement and evaluate this implementation by comparison to other well known and used IoT-related protocols, such as, HTTP, UDP, MQTT and CoAP.
This work explores the use of the QUIC protocol in a real-world scenario. This scenario will be the RuralTHINGS project, which focuses on creating an intelligent system for monitoring health-hazardous gases, such as radon gas and carbon dioxide, in residential areas. The main objectives are to discover the advantages and disadvantages of using the QUIC protocol, the scenarios in which it should be used, and how it compares to other similar protocols.

Keywords

QUIC, GQUIC, MQTT, CoAP, IoT, RuraLTHINGS, TCP, UDP

Contents

List of Figures

List of Acronyms

CoAP	Constrained Application Protocol
ECMP	Equal-cost multi-path routing
GRNET	Greek Research and Technology Network
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
MQTT	Message Queuing Telemetry Transport
NPM	Node Package Manager
OTT	Over-the-Top
RFC	Request for Comments
RTT	Round Trip Time
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UBI	Universidade da Beira Interior

Chapter 1

Introduction

1.1 Focus

With the IoT becoming increasingly prevalent in our lives, the demands on this technology are growing. Due to this, IoT systems need to become faster, more secure, and reliable, which has become challenging due to various limitations in current systems. One of these limitations lies in the transport layer protocols currently used, primarily TCP and UDP. While these protocols can establish communication between devices, they may not be effective for all types of connections. This leads to the need for the implementation of new transport protocols like QUIC.
Despite being initially designed to overcome TCP's limitations and inefficiencies, QUIC has quickly been adopted to work with the HTTP protocol. Currently, this protocol is being extensively researched for use in IoT, as its characteristics can address certain limitations in transport protocols when used in IoT, such as head-of-line blocking, connection migration, and high latency. IoT scenarios like autonomous vehicles, smart cities, and connected medical devices are among those that could benefit significantly from the use of QUIC.
This dissertation explores the application of QUIC, for IoT monitoring solutions, in a real-world scenario. Various measurements will be taken in this scenario using QUIC, and similar measurements will be conducted for other transport layer protocols, such as, HTTP, UDP, MQTT and CoAP. These measurements will serve as comparison values to determine if QUIC is indeed more effective.
In the first part of this dissertation, we delve into a detailed analysis of QUIC while searching for potential advantages and disadvantages that IoT may experience in its use. Other studies applying QUIC in IoT are also explored to find ways to assess QUIC's performance. Finally, we analyze a potential implementation of tests within the RuraLTHINGS project.

1.2 Main Objectives

The main research and development objectives of this dissertation are:

1. To evaluate the effectiveness of the QUIC protocol in a real IoT environment;
2. To compare the QUIC protocol with other protocols used in IoT;
3. To analyze the limitations of current transport protocols (TCP and UDP) in comparison to QUIC;
4. To highlight the situations for which the QUIC protocol can be used in IoT;

1.3 Dissertation Structure

The present document is organized as follows:

- The first chapter - Introduction - introduces the topic to be studied, the objectives, and the organization of the document;
- The second chapter - State of the Art - discusses QUIC and the transport protocols with which it will be compared (TCP and UDP). The Internet of Things and the advantages it can bring to it are also studied. Finally, other similar studies to those that will be conducted are analyzed.
- The third chapter - Implementation - starts by analyzing the possibilities we have for applying the QUIC protocol in IoT, and after that, it demonstrates the planning and subsequent implementation of it, thus allowing the collection of data for analysis.
- The fourth chapter - Discussion of Results - presents the analysis of the data obtained in the previous chapter, attempting to understand the reasons behind the results and also to identify the advantages and disadvantages of using the QUIC protocol.
- The fifth chapter - Conclusion - provides a summary of the most important results obtained in this study, as well as potential future work.

Chapter 2

State of the art

In this chapter, the functioning of the transport layer and the protocols that belong to it were investigated. An extensive study was also conducted on the capabilities of QUIC and its differences compared to other transport layer protocols. The Internet of Things was also thoroughly investigated, along with some widely used protocols within it, the possible benefits of using QUIC in the Internet of Things were also explored. Finally, some studies similar to the one this research aims to conduct were analyzed.

2.1 Communication Protocols and Transport Layer

In this section, the TCP/IP model was explored, concentrating mainly on the transport layer.

The TCP/IP model is the name given to a network architecture that serves as the basis for data transmission on the internet and other computer networks. This model is composed of four layers, each of which performs a specific function. The network access layer is the lowest layer of the model and allows data transmission through physical devices such as Ethernet cables and Wi-Fi. The internet layer is responsible for defining how data is transmitted within the network and between networks, using the IP protocol. The transport layer ensures that data is delivered an that it is reliable, handling retransmission in case of error. Finally, the last layer is the application layer, which allows users to interact with applications and where data is prepared (e.g., encrypted and formatted) to be passed to the other layers. The image 2.1 represents the model and some of the protocols it uses.[tcp]

The communication technologies within the internet have changed drastically since the early implementations of the TCP/IP protocol suite, and this change is due to the constant demand for web, mobile, and multimedia traffic.Various technologies and protocols have been created and improved to meet this demand, such as the HTTP/2 protocol and 5G, but continuous advancements are constantly sought to ensure that current technologies, such as virtual reality and augmented reality, can be reliable and efficient.

The transport layer is a crucial component when it comes to internet communication, as it is responsible for establishing the connection between two devices regardless of the application used or the existing networks between them. However, developing new efficient protocols at this layer becomes challenging due to issues such as middleboxes, socket Application Programming Interfaces, implementation of transport functionalities in the kernel, among others. This leads developers of new technologies to often rely on widely adopted protocols, namely TCP and UDP, even though these may not necessarily be the best protocols to use.[PFR+17][SRS+23]

TCP/IP Model

Layer	Protocol
Application	COaP MQTT HTTP
Transport	TCP UDP QUIC
Network	IPv4 IPv6
Data Link	Ethernet Wireless LAN
Physical	10 Base T 802.11

Figure 2.1: TCP/IP Model

2.2 What is QUIC?

In this section, we analyzed the history of the QUIC protocol.

QUIC is a protocol that operates at the transport layer (Layer 4 in the OSI model) and uses UDP. This protocol was created to enhance the communication flow of HTTP and make it more secure by utilizing UDP instead of TCP. It also establishes security and authentication for communication, eliminating the need to use the TLS protocol.

It was developed in 2012 by Google but was only announced in 2013. In 2015, it was submitted to the IETF, and in 2021, it was standardized by the IETF in RFC 9000. [rfc]

Currently, QUIC is being used, as HTTP/3, in several web browsers such as Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge. We can see a comparison between HTTP/3 and HTTP/2 in the image 2.2. Some Google services also employ this protocol, including YouTube and Google Drive. Many OTT companies, such as Netflix, Snapchat, Instagram, and Facebook, have also adopted QUIC. Facebook, in particular, has a usage rate of 75% of its traffic running over QUIC.[FBe20][auv]

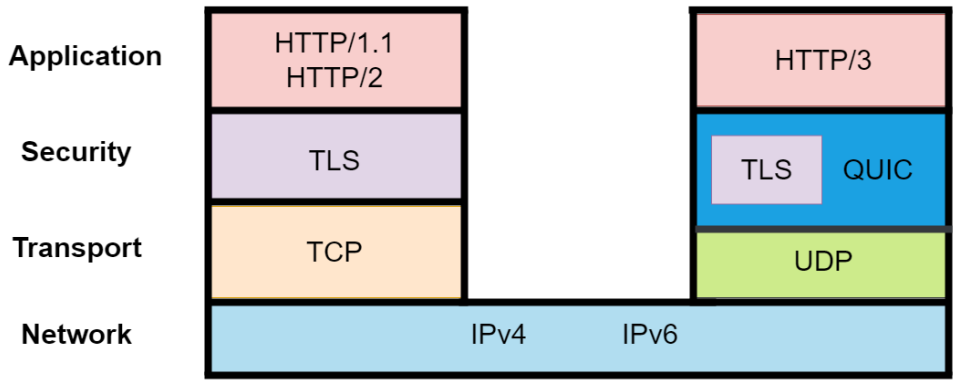

Figure 2.2: Layers in HTTP/2 and HTTP/3

2.3 How does QUIC work?

In this section we explain the QUIC protocol in comparison with the TCP protocol. We also describe the main features of QUIC.
In TCP communication, a three-way handshake is required. Each time a client needs to communicate with a server, there must be three distinct connections, namely:[tlsb][han]

1. SYN(Synchronize): A client sends a packet to the server to establish communication. This packet includes a sequence number with a random value.

2. SYN+ACK (Synchronize + Acknowledge): In response to the SYN, the server will send a packet that continues the communication and confirms the received packet. This packet will also include the sequence number received from the client plus 1, as well as a new sequence number for this packet.

3. ACK (Acknowledge): Upon receiving the server's response, the client will send an acknowledgment to the server. Along with the acknowledgment, the client will send the sequence number created by the server plus 1.

After the TCP communication is completed, we have a connection channel that links the client and the server, but this channel is not secure. Therefore, we often use the TLS (Transport Layer Security) protocol to ensure that we have a secure communication line (a connection that includes encryption, authentication, and integrity) with the server. For this purpose, four communications are required, which are:[tlsb][tlsa]

1. ClienteHello: A packet is sent from the client to the server requesting data to establish a secure connection and providing a list of the client's capabilities (TLS version, encryption algorithms).

2. ServerHello: In response to the client, the server will send a packet confirming the receipt of the ClientHello, selecting the capabilities provided by the client, and supplying data such as a digital certificate and cryptographic information necessary for the key exchange.

3. ChangeCipherSpec/Finished cliente: With the packet received from the server, the client will send a final packet containing the data for creating the shared session key between the client and the server. This key will be used for secure communication between the two. The packet will also include a message indicating that the client is ready to initiate communication.

4. ChangeCipherSpec/Finished servidor: Finally, upon receiving the final response from the client, the server will send a final packet enabling the client to obtain the session key. This packet will also inform the client that the server is ready to initiate communication.

We can better visualize this comunication in the image2.3.

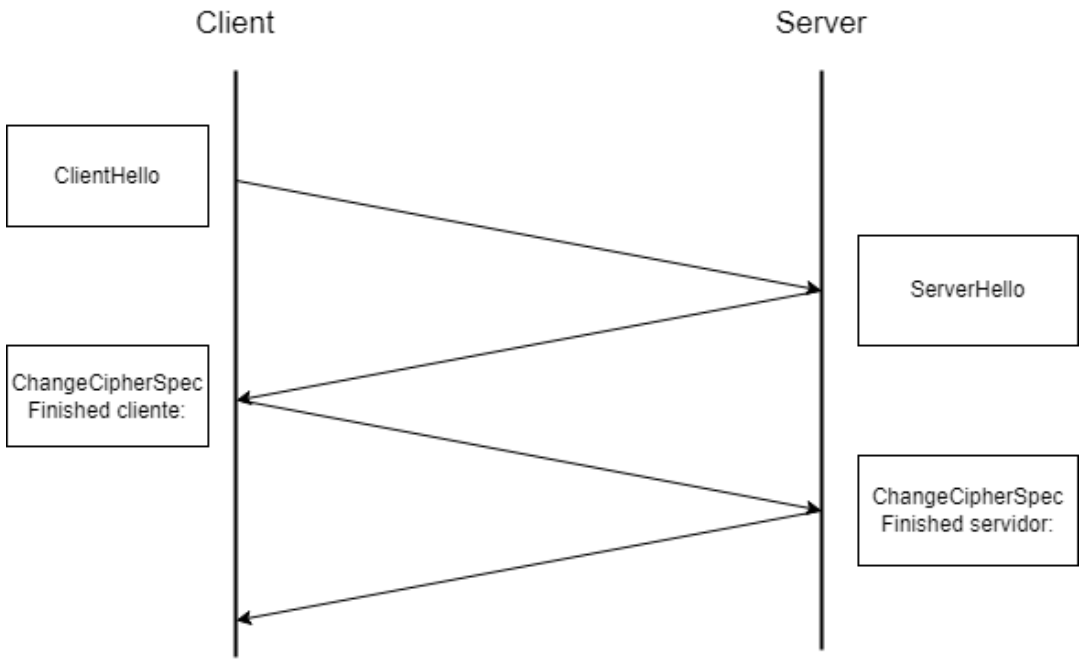

Figure 2.3: TCP + TLS and QUIC handshake

With this, we can observe that to establish a connection using the TCP protocol between a client and a server, three packets are required. To make this connection secure using the TLS protocol, an additional four packets need to be exchanged. It is possible to simultaneously send the last packet of the TCP protocol and the first packet of the TLS protocol. In total, six packets will need to be exchanged to initiate communication between two devices.

QUIC uses the UDP protocol, and as a result, it does not require a three-way handshake. This makes it possible to initiate information exchange immediately as we can see in the image 2.4. With this, the initial handshake can already include information that would typically be passed only after its completion. This means that with the first two exchanged packets, we can obtain the necessary configurations to start the connection, as well as exchange certificates and keys needed for a secure message exchange. The security of this protocol is based on TLS 1.3, incorporating many of the security principles and features introduced in it.

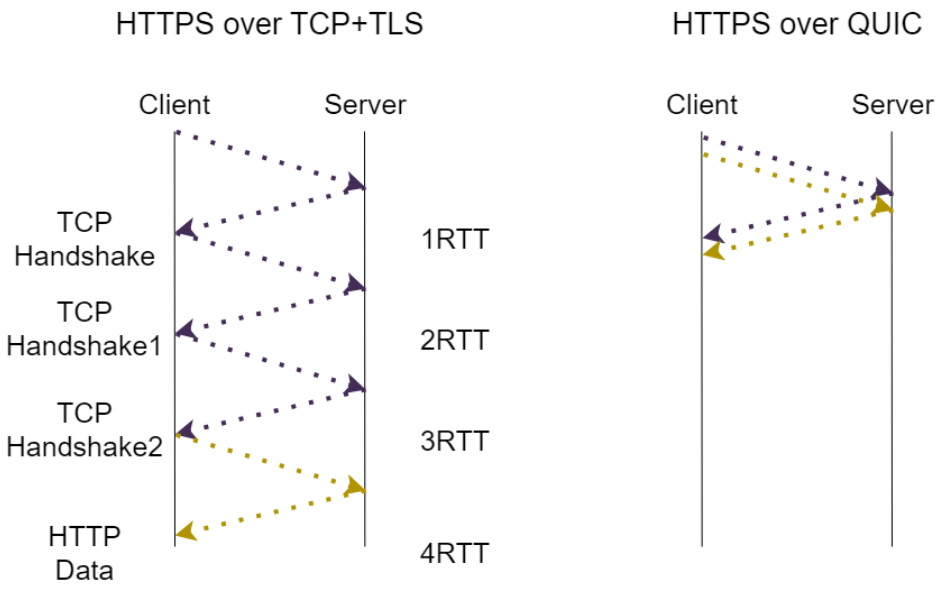

Figure 2.4: TCP + TLS and QUIC handshake

QUIC uses a combination of encryption and multiplexing to enhance security and data transfer. This allows for multiple packet transfers simultaneously without blocking the packet flow in case of a lost packet (head-of-line blocking). Additionally, QUIC incorporates features such as congestion control and flow control, further improving latency reduction and increasing data transfer speed.

Regarding header compression, QUIC employs a format called QPACK, which is a variation of the existing HPACK. It is designed to reduce the amount of redundant information transmitted, thus helping to prevent head-of-line blocking. This capability enables QUIC to transmit more data than TCP in low-capacity networks.

Using the features of this protocol, we can also achieve 0-RTT (Round Trip Time). This feature involves the ability to resume a previously established connection between the client and the server, using information from stored previous connections. This eliminates the need to renegotiate parameters such as session keys. While useful, this feature can pose security problems like replay attacks, where an attacker can replay the same connection multiple times. This issue can be mitigated by incorporating mechanisms such as adding extra information in the 0-RTT communication header. In the case of web applications, using 0-RTT only for GET requests, which are supposed to be idempotent (i.e., not altering the server's state), can also be a strategy. Another way to reduce risks is to limit the maximum number of requests per 0-RTT or set a time limit for their use. Although these issues are predominantly addressed in the context of web applications, they may need reconsideration when applied to IoT (Internet of Things) scenarios.

Another advantage of using the QUIC protocol is connection migration. The TCP protocol is tied to the traditional 4-tuple, consisting of the source IP address and port, and the destination IP address and port. If any of these elements change, the connection needs to be reestablished. This can occur when a user switches from a WiFi connection to a mobile network. In many situations, this can be detrimental to the client, as it leads to a delayed response time

due to the expiration and reestablishment of the connection. This issue can have significant impacts on IoT systems that need to remain mobile while transmitting real-time data, such as in autonomous vehicle driving. With QUIC, this problem can be addressed because connections are maintained by a 64-bit connection ID, allowing the connection to persist as long as the connection ID remains the same. This connection migration is possible in QUIC due to the use of the UDP protocol and its ability to operate without a fixed connection. However, due to the use of Equal-Cost Multi-Path Routing (ECMP) and anycast addressing employed by network operators, a single IP address may identify multiple servers. Since routers in the network may not handle QUIC properly, they might inadvertently send information to a different server than intended. This is because a UDP packet from the same connection may have different 4-tuple values, leading to connection interruptions.
An existing issue with QUIC is related to firewalls being unable to inspect QUIC packets, which can pose a significant problem. Malicious packets may go undetected when using this protocol. Consequently, it is common for some firewalls to block packets utilizing QUIC, leading to a fallback to the regular TCP protocol.[auv][roa18][cdn23][goo][ort17][ort19][JFD+22]

2.4 Internet of things

In this section, the Internet of things was explored, understanding the necessities that it has. The Internet of Things (IoT) is the capability of various physical devices, that incorporate sensors, software, and other technologies, to have the ability to collect data and exchange it over the internet. It is a technology that is constantly revolutionizing the way we interact with the environment around us, transforming it to be more efficient, useful, and comfortable. This technology has been developed due to constant advancements in areas such as communication protocols, wireless communications, and cloud computing, making it possible to achieve 15.14 billion connected IoT devices by 2023.
Currently, IoT devices can be found in various environments and serve different purposes. We can identify them in everyday objects such as smartwatches or devices found in our homes, including security and surveillance devices, household appliances, energy control systems, voice assistants, among others. These devices are also present in manufacturing industries, where they help gather information for subsequent analysis to assist companies in reducing costs and improving the manufacturing process, among other benefits. Smart cities also rely on IoT devices to collect data for effective resource management. Some examples of areas where the use of this data aids in city management include energy efficiency, traffic management, public safety, waste management, water supply network, sanitation, among others.
Initially, IoT relied on sparse and small messages for communication, but as it evolved, it has become the opposite. Messages started to be transferred with large amounts of information due to becoming more secure and complex, and they are increasingly sent in massive quantities to keep the data accurate and up-to-date. As a result, the demand for ways to make IoT faster, more precise, and reliable is beginning to increase.
Another challenge is the need for IoT to cover a wide range of scenarios that often differ in pri-

orities, typically involving latency and trust. Services like big data and artificial intelligence require high reliability, as these data will be used for predictions and providing trustworthy information to users. In contrast, monitoring systems prioritize low latency because they require data promptly. Medical and remote or automatic driving services require both fast and reliable data, as a lack of information or even errors in them can lead to significant harm to an individual's physical integrity. We can better analyse this priorities in the image 2.5.

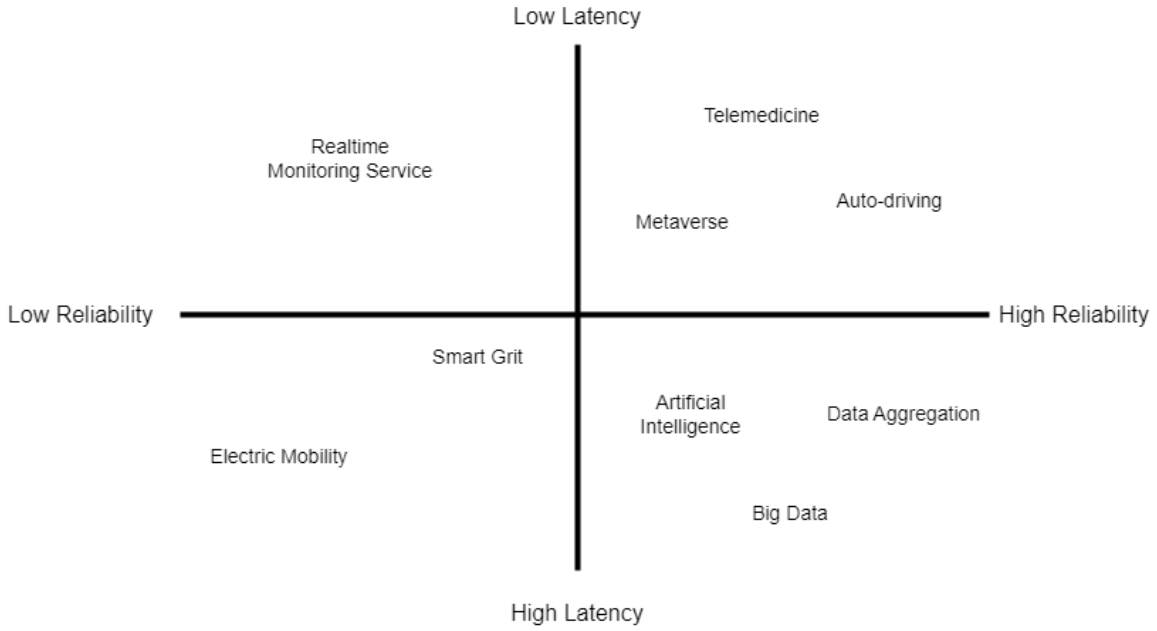

Figure 2.5: IoT priorities

With these demands that IoT is facing, the idea of integrating the existing QUIC protocol has emerged. Although initially designed for HTTP/3, this protocol can also be used in IoT, where its characteristics can enhance the utilization of application-layer protocols such as CoAP[SHB14] and MQTT[BG15]. [JNCK23] [iot23]

2.5 MQTT

In this section, we analyze a famous IoT protocol that uses TCP, which is MQTT.
MQTT is a protocol that operates at the application layer, featuring extremely lightweight messages and designed to connect devices in networks with limited bandwidth or high latency. It gained popularity in the Internet of Things (IoT) due to its ease of implementation, efficiency, and client dissociation. It is intended to operate over a transport layer protocol that provides a lossless, bidirectional, and ordered connection, typically associated with the TCP protocol.
The MQTT protocol operates on a Publish/Subscribe model. In contrast to the client-server model, this model separates the client sending the message (Publisher) from the client receiving the message (Subscriber). This means that the Publisher and Subscriber do not need to establish a direct connection; instead, an MQTT broker is situated between them, estab-

lishing the connection. The broker receives information created by the publisher, filters and categorizes this information by topic, and then sends it to the appropriate subscriber chosen based on the topic.
This protocol requires minimal resources, making it suitable for use on small microcontrollers with low power consumption. It is well-established in the IoT world and is compatible with a wide range of devices. Despite the advantages of this protocol, using TCP with MQTT introduces some drawbacks that could be addressed by employing the QUIC protocol. The initial handshake between the client and server in MQTT involves multiple packet exchanges, leading to increased latency and decreased connection setup time. This can pose issues in networks that may frequently disconnect, but these problems can be swiftly resolved with the use of QUIC. With 0-RTT, the handshake doesn't need to be repeated, significantly speeding up communication establishment. The use of TLS for connection security also increases latency, a concern not present in QUIC due to its built-in communication security in the initial connections.
Head-of-line blocking becomes a significant problem in MQTT communication, which is resolved in QUIC through multiplexing, reducing latency. This enables each data flow to have a different MQTT topic, and the ability to prioritize MQTT topics, thereby enhancing performance. In networks with high packet loss and disorder, TLS struggles to achieve good connection speed. However, QUIC, despite some reduction in speed, outperforms TLS in such scenarios.
Despite the advantages offered by QUIC, there are cases where TCP may be preferred or even required. This is because QUIC is not yet fully prepared or integrated for use in certain IoT devices. Additionally, QUIC incurs additional overhead due to its embedded encryption, which may complicate implementation on IoT devices with limited resources. [JFD+22] [MQT22]

2.6 CoAP

In this section, we analyze a famous IoT protocol that uses UDP, which is CoAP.
The CoAP is a protocol belonging to the application layer in the OSI model (Open Systems Interconnection). It was designed for use in networks of devices with limited resources, such as those found in IoT devices. This protocol is characterized by being lightweight and resource-efficient, making it suitable for devices with energy, bandwidth, and processing constraints. CoAP operates based on the request-response communication model, similar to the one used in HTTP. In this model, CoAP sends requests to the server, and the server responds with the requested data. These requests are the same as those found in the HTTP protocol, namely GET, PUT, POST, and DELETE. It is important to note that CoAP primarily uses UDP, which aids in increased data transfer and speed, as UDP is a connectionless protocol with lower overhead.
This protocol also allows for multicast communication, giving CoAP the ability to send a message to multiple receivers simultaneously, reducing network traffic and enabling more efficient data dissemination. Proxy support also provides scalability and system optimization,

contributing to greater efficiency.
The use of the QUIC protocol with the CoAP protocol brings with it the previously discussed advantages, such as the use of multiplexing to resolve head-of-line blocking, integrated encryption, connection migration for faster reconnection, among others. However, there are also disadvantages that may occur with the use of QUIC, such as the additional overhead it presents and the complexity of the protocol itself, which can lead to efficiency issues in devices with limited resources often used in conjunction with CoAP. The difficulty of implementing QUIC in certain networks can also lead to problems that would not arise if UDP or TCP protocols were used. [wha23] [JNCK23]

2.7 Related studies

In this section we search for the benefits of utilizing the QUIC protocol in IoT. Similar studies to this one were also analysed in this section so we could better comprehend the methods to apply the QUIC protocol in IOT and the results it could give us.
Within IoT, QUIC can bring benefits to a wide range of systems due to the discussed advantages. In the following points, we present some areas in IoT where the use of QUIC can lead to significant improvements:

- Connected Vehicles (V2X - Vehicle-to-Everything): V2X involves communication between a vehicle and any entity that can affect or be affected by the vehicle. Connected vehicles can greatly benefit from QUIC due to its quick connection and connection migration, allowing the vehicle to maintain a constant connection with road infrastructures, thereby enhancing efficiency and vehicle safety.[ALA+24]

- Smart cities: Smart cities use technologies to enhance services and the quality of life, utilizing data from internet-connected sensors for real-time information. This data helps cities make decisions to improve user experiences, such as optimizing traffic flow, reducing energy consumption, enhancing public transportation, crime detection, among others. The use of QUIC brings significant advantages to cities, providing rapid connection and improved data transfer through multiplexing, making the obtained data available much more quickly. Mobile sensors, including those used in public transportation and traffic management, also benefit from QUIC's connection migration feature.[IGK+23]

- Health and Connected Medical Devices: IoT medical devices, such as remote diagnostic equipment or personal health devices, can benefit from the QUIC protocol with its fast and secure connections, enabling efficient transmission of sensor data. This brings advantages that can contribute to the health of users of these medical devices. For example, in wearable heart monitoring devices carried throughout the day, QUIC's connection migration feature is particularly beneficial.

- Smart Agriculture: Sensors for monitoring agricultural crops and automatic irrigation systems require fast and reliable communication to analyze and maintain crops in

perfect condition. These sensors may also include weather monitoring sensors where quick decisions are needed in situations where weather conditions can damage crops. The use of QUIC enables these agricultural systems to communicate quickly and reliably, reducing potential losses in agricultural yields.[IGK+23]

Several studies on the QUIC protocol regarding its effectiveness and security have been conducted, but these studies have mainly focused on its use in web applications. Studies on the use of QUIC in IoT are more recent and primarily concentrate on analyzing the performance of QUIC compared to IoT application layer protocols. Some of these works have been reviewed to gather information on the obtained results. The following points provide observations from these studies:

- In [ALM22], the authors present the use of a gateway between an IoT device and a server was analyzed. This gateway was employed to establish a connection using the QUIC protocol, even in cases where the server did not support a direct connection with QUIC. The study utilized the MQTT protocol in conjunction with TLS (1.3) to establish the connection from the gateway to a cloud server. On the other side of the connection, where the IoT device connects to the gateway, the MQTT protocol was again used, but this time with QUIC. Throughout the study, data related to latency was analyzed, considering variables such as the use of TCP/TLS or QUIC, the presence of the gateway in communication, and the choice between a Full-Handshake or 0-RTT. Latency measurements were taken directly at the client, gateway, and server. The results obtained in this study were positive for the use of QUIC. They demonstrated that QUIC has 80% less latency for the client and 70% less for the broker compared to using TCP/TLS. Based on these results, the study concluded that, as expected, QUIC can be considered a good alternative to the TCP/TLS protocol.

- In [JNCK23] the authors explores the utilization of the QUIC protocol as an alternative to currently used IoT protocols. The study involved multiple connections between a client and a server, using both the CoAP protocol and QUIC. These connections were aggregated using two proxies, one for the client and another for the server. It's essential to note that the use of proxies was implemented to enable the use of QUIC without the need to modify existing applications on the client and server, which might not yet support QUIC.

  The study also tested the application of compression during the mapping of CoAP messages to the connection using QUIC, aiming to eliminate redundant traffic. The results obtained from this study revealed an 80% improvement in round-trip time when QUIC was used compared to other candidates. It was also concluded that the utilization of CoAP compression over QUIC improved performance by 10% compared to the experiment without compression.

- In [FZG+20], the usage of the QUIC protocol over MQTT was tested by the author, utilizing an implementation based on the GO programming language. For comparison, the same methodology was applied, but TCP/TLS was used instead of QUIC. To simulate various environments where this experiment could be applicable, Linux containers were employed, and multiple wireless networks were emulated using the ns3 simulator. One of these containers simulated the client as both a publisher and a subscriber, while another separate container simulated the server as the Broker.

  The obtained results indicated that QUIC exhibited an improvement in response time compared to the TCP protocol. However, these results were unexpected, considering the anticipation that the 0-RTT feature used by QUIC would be much faster. To conclude, the study discussed that the issue might be related to the server using QUIC being unable to read packets coming from 0-RTT, thus necessitating retransmission using 1-RTT.

- In [Her21], the study involves creating an analytical model to estimate parameters causing degradation in data transmitted by sensors in CoAP. These parameters include packet loss and latency. The model is subsequently validated through experimental frameworks, where the advantages of the QUIC protocol are tested in comparison to UDP and TCP over CoAP.

  The results obtained indicate that under the same tested conditions and using the analytical model described in the study, QUIC experiences fewer losses than TCP and UDP. The analytical model's results estimate a packet loss probability of 1.67% for UDP, 65.06% for TCP, and 79.42% for QUIC based on experimental outcomes. It's important to note that the Round-Trip Time (RTT) is assumed to be constant during the experiment, potentially explaining the lower performance of TCP compared to QUIC.

Other studies related to QUIC that are not directly associated with IoT were also analyzed, such as [MKM16]. In this study, the performance of QUIC was explored in comparison to HTTP 1.1 and SPDY. The conclusion drawn was that in 40% of the tested scenarios, QUIC improved the webpage loading time. It was also noted that QUIC could provide benefits to mobile internet traffic, particularly in scenarios with high Round-Trip Time (RTT). However, there could be performance issues with QUIC due to limitations in UDP traffic that some networks impose. Additionally, studies related to the security of QUIC and its performance in the presence of attackers, like [LJBNR15], were examined. An important conclusion from this study highlighted the possibility that QUIC might not be able to use the 0-RTT connection, thus being forced to redo the handshake, which could lead to increased latency.

2.8 Findings and Conlcusions

Ref / Nome	Protocols Used	Metrics Used	QUIC Used
[ALM22] Analyzing the Latency of QUIC over an IoT Gateway	MQTT / QUIC / TLS	Latency	Pure Go
[JNCK23] Use of QUIC for CoAP transport in IoT networks	UDP / TCP / SCTP / QUIC	Number of clients / Number of packets exchanged / RTT / Goodput	quic-go
[FZG+20] And QUIC meets IoT: performance assessment of MQTT over QUIC	MQTT /TCP / TLS / QUIC	completion ratio / RTT after connection	quic-go
[Her21] Analysis of QUIC Transported CoAP	CoAP / UDP / TCP / QUIC	Message Loss probability / Loss burstiness	Not Mentioned
This Study	CoAP / MQTT / HTTP / QUIC	Number of packets send / Packet Loss / RTT	node-quic

Table 2.1: Summary of protocols and metrics used in the analysed studies

In this chapter, we explore how QUIC can minimize the issues we face with other protocols at the transport layer. We observe that the reduced number of exchanged messages and multiplexing can contribute to the effectiveness of QUIC as a transport protocol. Additionally, we note that utilizing 0-RTT can further enhance efficiency, although its implementation may not always be feasible and may also entail security costs. Furthermore, we recognize that efficient connection migration can be achieved when using QUIC, thanks to the use of a connection ID instead of the 4-tuple used by other protocols.
We also delve into the benefits of using QUIC in the Internet of Things (IoT), employing MQTT and CoAP as application layer protocols to better understand these advantages. The requirements that IoT technologies need to operate efficiently, as well as scenarios where QUIC would be beneficial in these technologies, are also explored.
Finally, various studies regarding the direct application of QUIC on application-layer protocols and its effectiveness in terms of response time and security are investigated. We can see some of the characteristics of each study compared to this one in the table2.1
These studies are crucial for identifying potential challenges that may arise during this study, as well as finding solutions. One such challenge is the possibility that the technologies used in the project for measurements may not support QUIC, resulting in an inability to observe the protocol. This can be addressed by introducing proxies or gateways at certain points in the connection to facilitate the exchange of messages between QUIC and other transport protocols.

Chapter 3

Implementation

This section presents the planing and implementation of this study. We began by discussing an initial implementation plan, using the RuralTHINGS project to analyze the protocol to be studied, and then proceeded to plan how it will be implemented. The technologies used in this implementation and their characteristics are also mentioned. Finally, we specify how the implementation was executed by detailing the steps taken.

3.1 RuraLTHINGS project

In this section, we understand the motives for the creation of the RuraLTHINGS project[rur]. We also present the architecture of this project in the image 3.1.

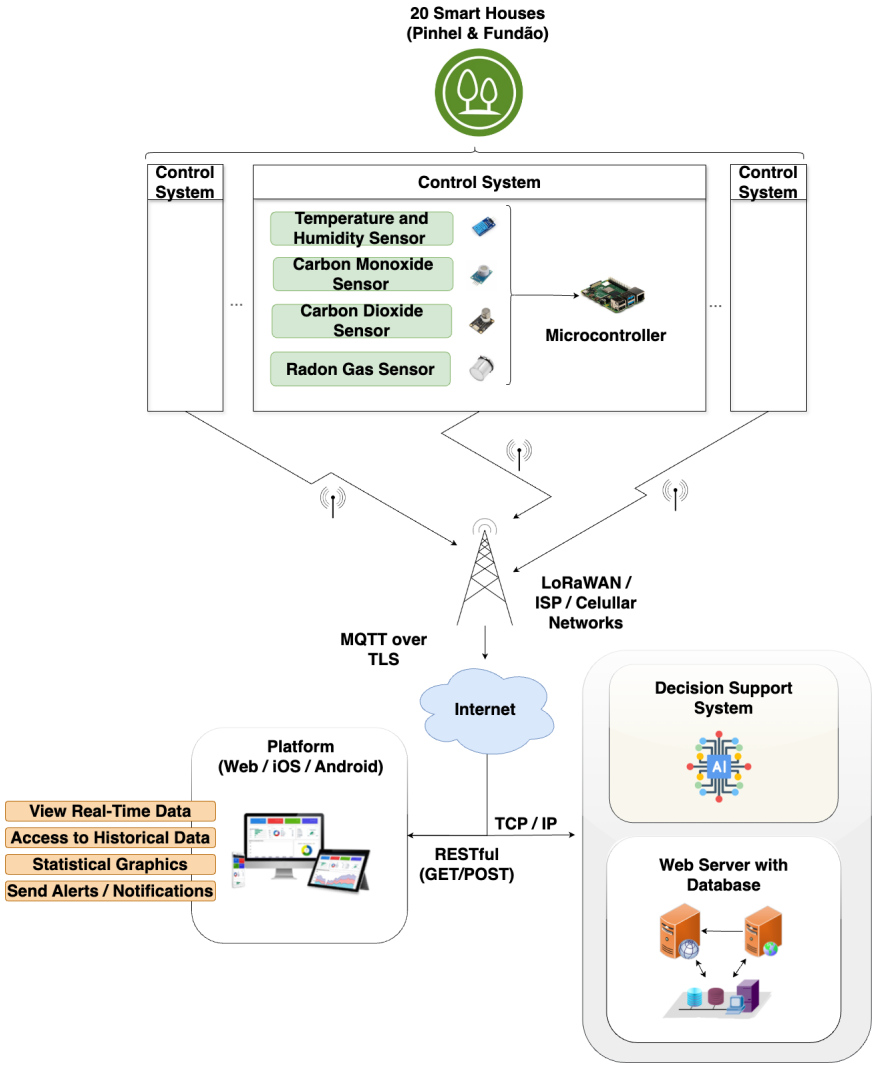

Figure 3.1: RuraLTHINGS System architecture

Radon gas is a radioactive gas that originates from the soil or rocks and tends to accumulate in enclosed spaces, especially in residential homes. This gas is highly dangerous when exposed to it indoors and can lead to health problems such as lung cancer. Despite the potential health hazards, radon gas cannot be easily detected through simple means, and the only reliable method of detection is through measurement. This means that inhabitants of a house could be exposed to this dangerous gas without being aware of it. The regions with higher concentrations of radon gas are mainly in the northern and central areas of the country.
The RuraLTHINGS [rur] project was created to develop an intelligent system for monitoring health and well-being in remote areas using IoT. It focuses on researching and developing an ecosystem of solutions based on the IoT paradigm, specifically applied in rural and remote areas. Real-time data will be monitored to identify the presence and quantity of gases that can be harmful to health. The gases targeted for detection include carbon monoxide, carbon dioxide, and radon gas. Additionally, temperature and humidity information will also be monitored.[Rur23] [rad23]

3.2 Implementation planning

In this section we talk about the planing involved on the implementation of the protocol that is being studied, and the problems that aroused during it. We also exhibited the architecture of this implementation in the image 3.2.
To analyze the effectiveness of QUIC in IoT, the initial plan for this research was to use the system created for the RuraLTHINGS project. In this project, carbon monoxide, carbon dioxide, radon gas, humidity, and temperature sensors were installed in 20 homes and connected to a microcontroller. These microcontrollers were then connected to a LoRa/ISP/mobile network, which sends the data collected by the sensors to a database. This data can later be viewed by the customer through a web application hosted on a server.
To evaluate QUIC's performance, the following metrics were used: average RTT and the number of packets. To compare the obtained data, the same tests would be conducted using the UDP and TCP protocols.
During the project, it was decided that Arduinos/ESP32 would be used to obtain data from the sensors. This data would be sent to a Raspberry Pi inside the home so it could be exported; this would use the Mosquitto MQTT broker to subscribe to the various topics (sensor data) that may exist.
Once the various Raspberry Pis in the homes receive the data, it will be transmitted to a central MQTT broker, which is the Amazon IoT Core, where the data is subsequently sent to an SQL database.
Due to the use of these technologies, the implementation and analysis of the QUIC protocol became unsuitable, as Amazon IoT Core does not allow the use of this protocol. This led to the search for other alternatives in an IoT environment to continue the development of this study.
It was quickly decided that the application of the Raspberry Pi in the system as a gateway would be the best option due to certain essential features such as:

- Ability to communicate via Wi-Fi and Bluetooth;
- Presence of I/O (Input/Output) interfaces for collecting data from sensors;
- Support for various programming languages and the Linux operating system;
- An active community and a large amount of documentation;
- High usage in low and medium-level IoT projects.

Regarding the server with which the Raspberry Pi will communicate, various technologies were analyzed, with the main characteristic required being the ability to use the QUIC protocol and the MQTT protocol. Among the various technologies researched, many were found to be either incapable of communicating with the QUIC protocol or vastly difficult to implement. Among the capable ones, two stood out: Node.js and the Go programming language. Go, using the quic-go library, was by far the most used in other studies, which, like this one, aim to analyze the effectiveness of this protocol. However, due to the native support for the QUIC protocol and to obtain more varied and innovative data, Node.js was used.
To obtain suitable data for testing a transport layer protocol, an internet connection is required. For this reason, a remote server was used, in this case, the okeanos global. The okeanos global was chosen mainly because it can provide a virtual machine with considerable capabilities at no cost. PuTTY was also used to establish an SSH connection with the cloud, WinSCP for transferring files between the local computer and the cloud, and Wireshark to observe the transferred packets and confirm that the correct protocols were being used.
The protocols to be tested were selected to allow for data comparison, thus providing a better understanding of the QUIC protocol. These protocols were chosen based on previously analyzed studies and those currently used in IoT. The HTTP protocol was the first selected due to being a well-known and widely used protocol today. The MQTT and CoAP protocols were also quickly chosen, with MQTT being one of the most used and recognized in IoT and employing a Publish/Subscriber model. CoAP was selected because it is a protocol that uses UDP, allowing for a closer comparison with the QUIC protocol, as it also uses UDP.
Finally, it was decided how the tests would be conducted. Based on research done on previous implementations of the QUIC protocol, it was concluded that it would be necessary to analyze several messages to obtain viable data. Therefore, it was decided that fifty messages would be sent in succession to gather more viable data.
To obtain data from more diverse perspectives, it was also decided that protocols using TCP should be tested in situations where maintaining a connection was impossible and required forming a new handshake. This was designed to demonstrate scenarios where sensors or gateways might be in motion, such as in the cases of smart vehicles or drones. As a result, the MQTT and HTTP protocols were tested in situations where the handshake was executed only once, as well as in situations where all communications required a new handshake.
These were repeated three times for each protocol, and in the end, those with the best results were selected, which in this case were the ones with the best average RTT. This was done to

account for possible internet connection disturbances, aiming to eliminate unwanted variations in the collected data that may arise due to its instabilities.

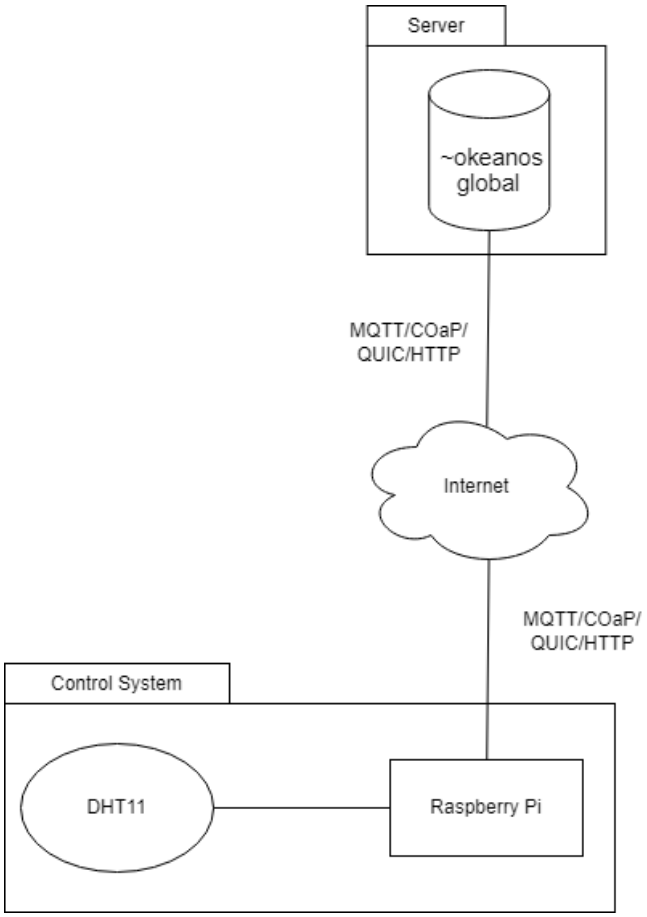

Figure 3.2: System architecture

3.3 Technologies

In this section we talk about the applied technologies in this study.

3.3.1 Node.js

Node.js is an open-source JavaScript runtime environment for server-side programming. It allows code execution outside the browser, enabling the creation of backend applications. It's also worth noting that it is built on Google's V8 engine, which allows JavaScript to be compiled directly into machine code (instructions that the processor interprets and executes). [nod23a] [nod23b]
Unlike many other programming languages and environments that rely on multi-threading, Node.js uses a single-thread model. This means that Node.js does not automatically create multiple threads to handle various operations. Instead, it processes operations asynchronously. This brings advantages such as simplicity and ease of code development, better resource utilization when performing I/O operations, improved memory efficiency, and easier debugging. However, there are also disadvantages, such as limited performance in

CPU-intensive tasks, dependence on asynchronous operations, and latency issues in heavy computational processing. [nod23c]
With Node.js, we also have access to NPM (Node Package Manager), which is a package management system for Node.js. It allows developers to easily share code.

3.3.2 okeanos global

The okeanos global is an ”IaaS” (Infrastructure as a Service) platform, created by GRNET (Greek Research and Technology Network). This platform was designed to be used by the academic and research community in various fields. It allows users to create virtual machines, private networks, and store data, making it an excellent platform for researchers and educators to run simulations, conduct data analysis, and develop collaborative projects. It is also worth noting that this is a non-profit platform offering free resources to users affiliated with universities and other educational institutions. [oke]

3.3.3 Raspberry Pi

The Raspberry Pi is a low-cost, small-sized microcomputer developed by a non-profit organization based in the United Kingdom. It was developed with the aim of encouraging basic computing education in schools and underdeveloped countries, and it quickly became popular among developers, educators, and enthusiasts due to its versatility and affordability. It is also widely used in IoT not only for the reasons mentioned earlier, but also for its ability to connect with various sensors and actuators. [rasb]

3.3.4 DHT11

Since it was necessary to gather environmental data, a sensor was required, and the sensor used was the DHT11. This sensor allows for reading both temperature and humidity in the environment, and it can then send the data to a device connected to it. It can read temperatures between 0°C and 50°C, and humidity levels between 20% and 90%, with an accuracy of ±1°C for temperature and ±1% for humidity. [dht]

3.3.5 Eclipse Mosquitto

Eclipse Mosquitto is a broker that uses the MQTT protocol and enables communication between connected devices (such as actuators, sensors, and servers), receiving and sending messages. It was created with the goal of occupying little space and requiring minimal computational resources to operate. It also offers security features such as TLS/SSL for data encryption. Another characteristic of this broker is the simplicity of installation and configuration, although it also offers more advanced functions depending on the application’s objective. [mosa][mosb]

3.3.6 Putty

Putty is an open-source terminal emulator that also allows file transfers. It supports various network protocols such as SSH, Telnet, SCP, among others. This program is often used to enable secure connections to remote servers and to execute commands remotely.

[put23]

3.3.7 WinSCP

WinSCP is an open-source tool that allows file transfer between a server and a local machine. It supports protocols such as SFTP, SCP, FTP, and WebDAV. Features like a simple and effective graphical interface, a session manager that allows saving connection information, file synchronization, and an integrated text editor make this program an excellent option for working with remote servers.

[win]

3.3.8 Wireshark

Wireshark is a packet sniffer, meaning it captures and analyzes packets on a network. This tool allows for real-time packet visualization, enabling detailed observation of the protocols in use. It also features an intuitive interface, various ways to filter and search for packets, data export for analysis, and the creation of graphs.[wir]

3.4 TestBed

As previously mentioned, the implementation of the RuraLTHINGS project was not feasible, and as a result, alternative approaches had to be explored. Given that the main objective was the application of the QUIC protocol in IoT, the search for alternatives focused on application layer protocols. Consequently, the use of various types of sensors became unnecessary, and thus, we used only one sensor. However, in an attempt to align the study with the RuraLTHINGS project, the use of the Raspberry Pi and the principle of sending data to a cloud platform were retained. This led to the use of the components we observe in architecture 3.2, which include the DHT11 sensor, the Raspberry Pi, and the Okeanos global server.

The Raspberry Pi used for this experiment was the Raspberry Pi 400, we can see this component characteristics in the table 3.1: [rasa]

Specs
Broadcom BCM2711 quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz
4GB LPDDR4-3200
Dual-band (2.4GHz and 5.0GHz) IEEE 802.11b/g/n/ac wireless LAN
Bluetooth 5.0, BLE
Gigabit Ethernet
2 × USB 3.0 and 1 × USB 2.0 ports
Horizontal 40-pin GPIO header
2 × micro HDMI® ports (supports up to 4Kp60)
MicroSD card slot for operating system and data storage
A compact keyboard
5V DC via USB connector
Operating temperature: 0°C to +50°C ambient
Maximum dimensions: 286 mm × 122 mm × 23 mm

Table 3.1: Raspberry Pi Specs

The okeanos global server has already been configured to obtain the best possible features, which we can see in the table 3.2:

Specs
4 CPU cores
4096 MB of memory
40 GB os system disk
Ubuntu as operative system

Table 3.2: Okeanos Global Specs

3.5 Implementation development

In this section, a detailed explanation of the procedures applied in this implementation has been described.

After deciding on the technologies to be used, we began to explore QUIC natively integrated into Node.js. This led to the discovery that the integration of QUIC in Node.js is still under development and is still referenced as experimental. After several attempts, the decision to use the experimental QUIC was abandoned due to the difficulty of implementation (both on the Raspberry Pi and on the cloud server) and the various existing failures in the program.

Wanting to maintain Node.js, libraries that would allow the use of QUIC were researched, with the only viable one being “node-quic.” Although old and without recent updates (the last one being 6 years ago), this library proved to have what was necessary for the implementation of the QUIC protocol. However, this library does not use QUIC but rather GQUIC, which is a less developed and older version of the QUIC protocol. Despite this setback, after analyzing this protocol, it was concluded that using GQUIC would not considerably alter the results to be obtained, and therefore the use of the “node-quic” library was maintained.

Libraries for the application of the HTTP, MQTT, and CoAP protocols were also explored. These were easier to acquire and analyze since they are widely used by the Node.js community; the libraries obtained were ”http,” ”mqtt,” and ”coap.”

To use MQTT, it was also necessary to use an MQTT broker, which in this case was the

”mosquitto” broker. This broker was chosen due to its simple configuration and high usage in the IoT community.
Initially, applications were developed to transmit data locally between these four protocols while calculating the RTT by measuring the time it took for the information to be sent and the ACK to be received. The RTT values were later saved in an xlsx file for further analysis and discussion using the ”xlsx” library.
After the development of the applications that created the data for analysis, a server was created in the cloud using okeanos global. The creation of a virtual machine was quite simple, requiring only a login with an account belonging to a university and choosing the characteristics of the virtual machine, which in this situation used the best settings that were allowed. PuTTY was used to execute an SSH connection that allowed us to start working with the virtual machine, and WinSCP was also used to transfer files to the server side.
After updating the server, installing Node.js and its packages, and installing and configuring the MQTT broker ”mosquitto,” the files that created the servers for the protocols to be tested were transferred. Communication between the local computer and the server was also tested to verify if the results were as expected. It is worth noting that both the MQTT broker and the server were running on the same server, which did not lead to any alteration of the results since the communication being tested was between the broker and the gateway.
Having the server ready to communicate, we move on to the installation of a sensor on the Raspberry Pi and its configuration. To create a simulation closer to the reality of IoT, a DHT11 sensor was used to measure temperature and humidity in the environment. It was chosen for being easy to use, inexpensive, and because it is one of the sensors that would be used in the RuraLTHINGS project. After installing the sensor and making the necessary configurations on the Raspberry Pi, Node.js and the required libraries were installed, and finally, the files needed for testing were transferred. It is important to mention that the applications made afterwards required some adjustments to obtain data from the DHT11 sensor, including the installation of a new library to make data reading more straightforward, which was “node-dht-sensor.”
After the adjustments were made, tests were conducted and values were obtained in different Excel files, one for each protocol. After the data was collected, tests were run again for 10% and 20% packet loss using the tc (traffic control) command on the server. Once all the data was gathered, it was entered into an Excel file where the averages of the RTT values were calculated and the respective graphs were created for data analysis.

3.6 conclusion

As we can observe in this implementation, the initial goal of using the RuraLTHINGS project to showcase the advantages of the QUIC protocol could not be put into practice due to the limitations of using QUIC with today’s technologies. This became a constant problem throughout the development of this investigation, leading to the search for various alternative implementations, which are still not fully refined. Nevertheless, these implementations provided enough possibilities to gather data for further study. We can also observe that Node.js proved

to be a very versatile tool due to its simplicity, large community, and diversity of libraries. The Raspberry Pi also proved to be a very capable tool for conducting studies in the IoT field, as expected. With the data obtained from this implementation, we proceeded with the analysis and discussion of these results. Initially, good results were expected compared to other protocols like HTTP, MQTT, and CoAP, with the latter being the closest to QUIC, as it also uses UDP. However, when packet loss was introduced, it was expected to progressively lose efficiency.

Due to the implementation of a non-native solution in Node.js and the use of GQUIC, the expected results became more unpredictable, potentially showing significant increases in RTT compared to other studies.

Chapter 4

Evaluation Results and Discussion

This chapter presents the evaluation results and its discussion. We present the data that was collected in the above-mentioned implementation chapter and analyse it. We also discuss the possible causes and features that led to the evaluation results.

4.1 Results

In this section, we can observe the tables with the results obtained from the evaluation of the above-described implementation.The tables show, in the first column (Nº), the number of messages sent, and in the first row, the protocols used in this study. With this, we can see all the messages that were sent and the respective RTT for each protocol used. In Table 4.1, we can observe the results when no induced packet loss is applied; in Table 4.2, the results with 10% packet loss; and finally, in Table 4.3, the results when a packet loss of 20% is induced. To better visualize this data, several Figures were also created. Figures 4.1, 4.2 and 4.3 present line graphs where the X-axis refers to the message number, while the Y-axis represents the RTT value in milliseconds (ms).

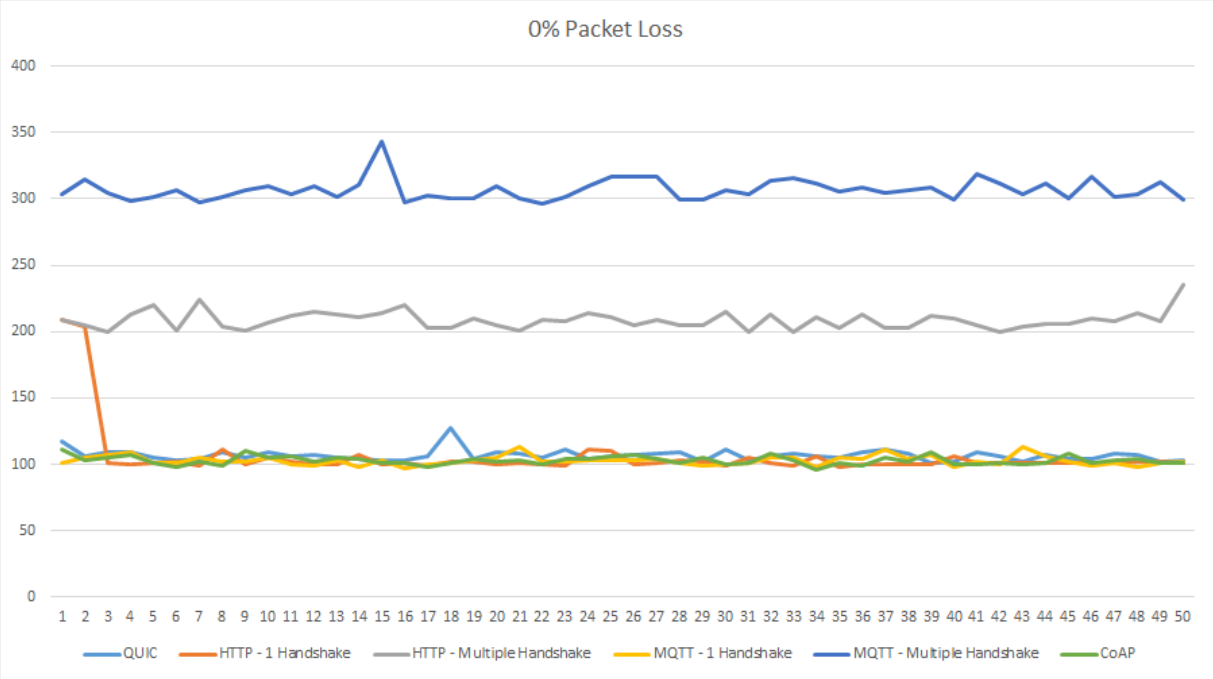

Figure 4.1: 0% Packet Loss

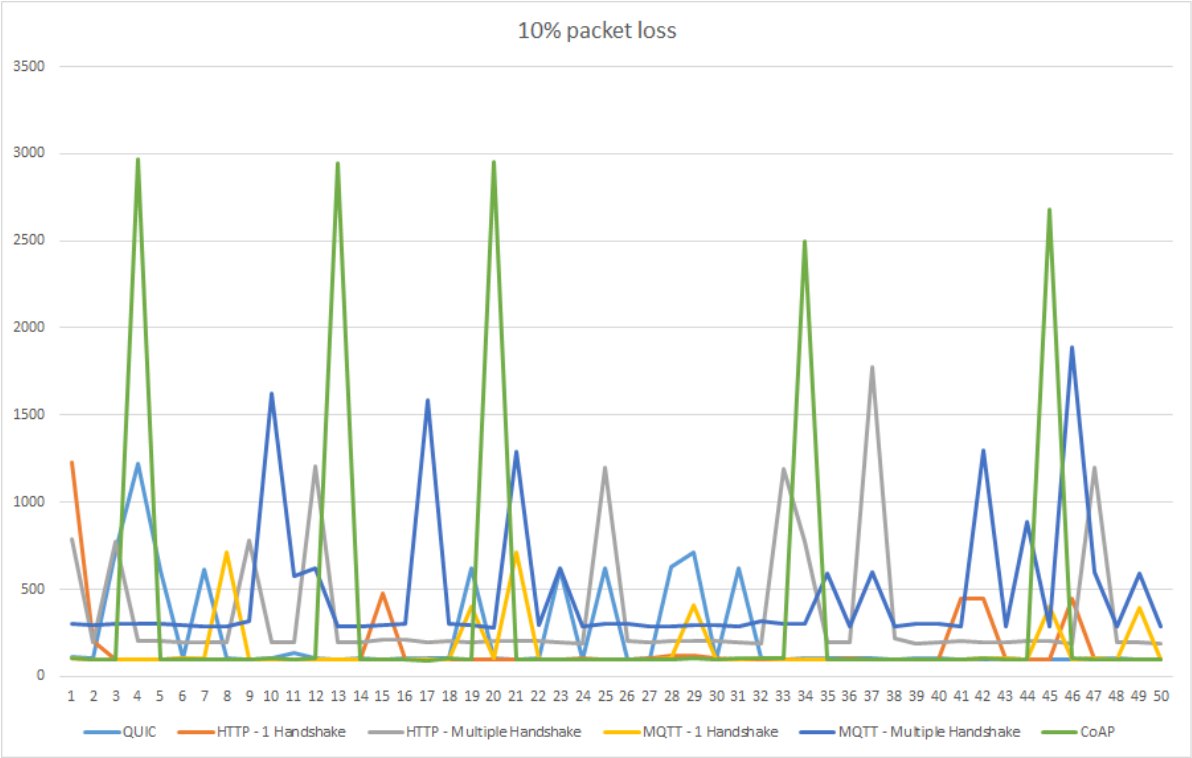

Figure 4.2: 10% Packet Loss

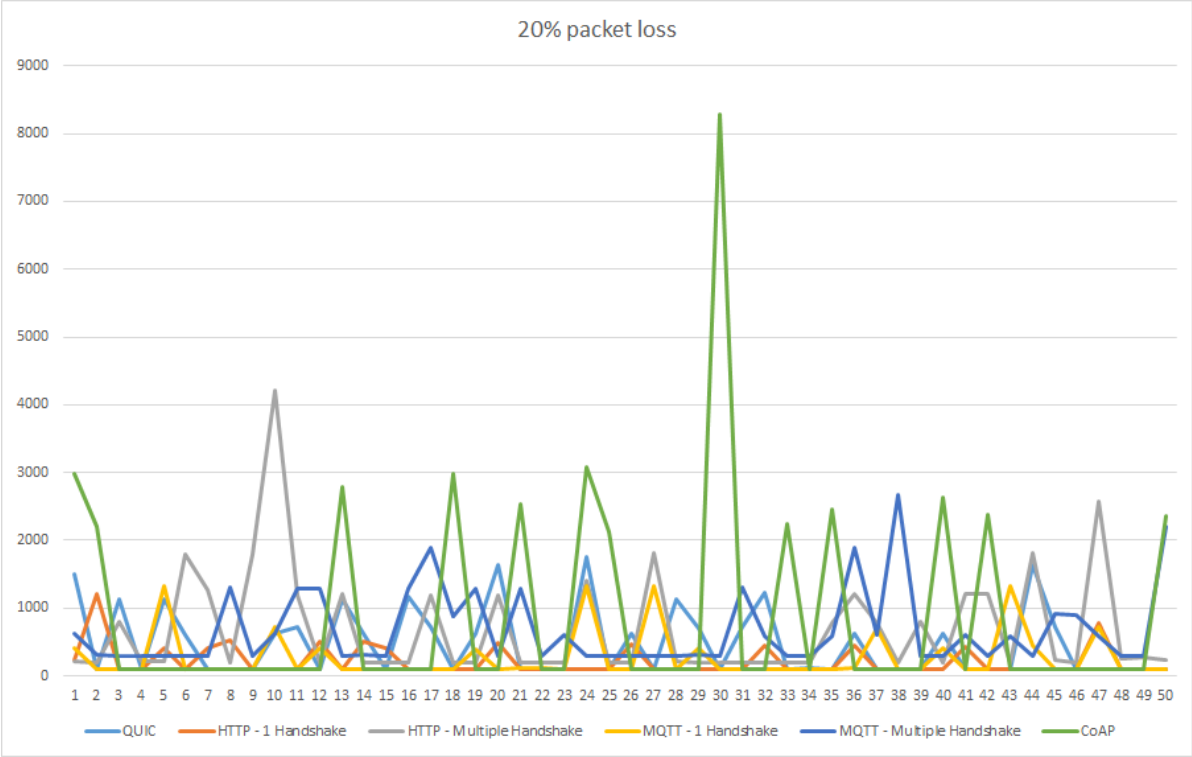

Figure 4.3: 20% Packet Loss

Nº	QUIC	HTTP - 1 Handshake	HTTP - Multiple Handshake	MQTT - 1 Handshake	MQTT - Multiple Handshake	CoAP
1	117	209	209	101	303	111
2	106	204	205	105	315	103
3	109	101	200	107	304	105
4	109	100	213	109	298	107
5	105	101	220	101	301	101
6	103	102	201	101	306	98
7	104	99	224	105	297	102
8	109	111	204	102	301	99
9	105	100	201	102	306	110
10	109	105	207	105	310	105
11	106	102	212	100	303	106
12	107	100	215	99	310	102
13	105	100	213	103	301	105
14	104	107	211	98	311	104
15	103	100	214	103	343	101
16	103	101	220	97	297	101
17	106	99	203	100	302	98
18	127	102	203	101	300	101
19	104	102	210	103	300	104
20	109	100	205	105	310	102
21	108	101	201	113	300	103
22	105	100	209	102	296	100
23	111	99	208	102	301	104
24	104	111	214	103	310	104
25	106	110	211	103	317	106
26	107	100	205	103	317	107
27	108	101	209	104	317	104
28	109	103	205	101	299	101
29	102	102	205	99	299	105
30	111	99	215	100	306	100
31	103	105	200	101	303	101
32	106	101	213	105	314	108
33	108	99	200	105	316	103
34	106	106	211	98	312	96
35	105	98	203	105	305	101
36	109	100	213	104	308	99
37	111	100	203	111	304	105
38	108	100	203	104	306	102
39	101	100	212	107	308	109
40	102	106	210	98	299	100
41	109	101	205	102	319	100
42	106	101	200	100	312	101
43	102	101	204	113	303	100
44	107	101	206	106	312	101
45	104	101	206	102	300	108
46	104	101	210	99	317	101
47	108	101	208	101	301	103
48	107	102	214	98	303	104
49	102	102	208	101	313	101
50	103	101	235	102	299	101

Table 4.1: Data - 0% Packet Loss

Nº	QUIC	HTTP - 1 Handshake	HTTP - Multiple Handshake	MQTT - 1 Handshake	MQTT - Multiple Handshake	CoAP
1	110	1226	791	103	304	107
2	106	204	193	100	298	96
3	718	101	772	99	306	99
4	1224	98	203	98	299	2970
5	608	99	200	99	301	98
6	102	106	199	102	295	101
7	617	98	194	106	288	100
8	102	98	196	716	288	101
9	100	97	780	99	314	97
10	107	98	195	99	1622	102
11	134	98	194	99	576	101
12	104	98	1204	101	625	102
13	98	101	197	100	287	2944
14	103	97	198	99	289	96
15	101	477	208	99	292	99
16	102	96	212	99	304	96
17	104	96	198	99	1589	93
18	103	96	202	99	299	105
19	624	97	195	398	295	97
20	103	97	202	105	282	2954
21	101	95	201	709	1292	99
22	108	97	200	99	291	98
23	617	98	196	99	618	97
24	98	104	189	100	284	101
25	620	98	1202	100	305	97
26	99	100	206	100	300	98
27	106	107	195	100	287	97
28	626	117	204	103	288	99
29	716	122	201	409	295	102
30	100	105	205	100	297	101
31	625	96	193	99	290	102
32	102	97	191	102	314	102
33	98	97	1194	100	301	102
34	102	96	772	95	302	2498
35	102	96	196	101	588	98
36	103	107	196	97	287	96
37	105	97	1776	99	598	97
38	99	99	222	98	285	97
39	106	96	191	101	305	96
40	107	97	198	100	305	101
41	98	449	205	100	289	95
42	100	444	194	103	1298	103
43	102	97	193	102	288	96
44	99	97	205	98	890	96
45	98	97	201	399	287	2682
46	98	446	191	99	1893	104
47	103	96	1198	102	600	98
48	102	94	197	98	288	97
49	100	96	193	397	593	96
50	99	97	191	99	289	96

Table 4.2: Data - 10% Packet Loss

Nº	QUIC	HTTP - 1 Handshake	HTTP - Multiple Handshake	MQTT - 1 Handshake	MQTT - Multiple Handshake	CoAP
1	1495	228	224	403	634	2981
2	102	1210	192	107	307	2194
3	1138	99	799	103	289	102
4	104	100	215	100	284	97
5	1137	405	218	1321	302	101
6	614	100	1787	99	293	99
7	105	409	1259	98	297	97
8	102	523	195	96	1307	98
9	101	98	1788	100	295	97
10	619	101	4206	714	622	102
11	716	97	1193	97	1289	98
12	106	513	195	401	1285	99
13	1135	96	1206	97	293	2797
14	622	512	196	97	306	99
15	100	413	194	98	298	100
16	1178	100	197	99	1288	100
17	714	94	1194	99	1886	93
18	106	101	194	99	876	2980
19	622	102	193	395	1291	102
20	1643	480	1191	100	284	97
21	106	103	194	113	1294	2538
22	101	102	192	109	294	98
23	103	96	192	101	599	96
24	1747	102	1409	1320	291	3075
25	101	97	204	99	286	2126
26	618	464	202	104	290	107
27	105	98	1816	1328	284	100
28	1132	105	216	98	289	100
29	712	97	196	413	312	97
30	104	102	201	99	297	8291
31	729	100	196	101	1309	101
32	1234	458	194	102	588	98
33	100	95	197	102	292	2240
34	123	100	199	100	292	101
35	99	96	774	99	584	2449
36	621	453	1204	111	1897	98
37	102	99	786	712	600	101
38	102	100	199	99	2667	93
39	101	99	800	101	290	104
40	621	102	203	404	291	2627
41	104	432	1201	99	599	97
42	105	101	1202	97	297	2382
43	107	96	202	1323	578	98
44	1644	99	1807	454	290	100
45	714	99	229	102	926	97
46	104	101	196	100	900	97
47	99	778	2565	714	581	96
48	104	105	263	102	296	100
49	101	98	272	107	303	104
50	98	101	239	105	2199	2361

Table 4.3: Data - 20% Packet Loss

In addition to the line graphs, bar charts were also generated where we can observe the average RTT for each protocol at various levels of induced packet loss. In charts 4.4,4.5,4.6 and

4.7, the Y-axis contains the protocols used, while the X-axis shows the average RTT. Chart 4.4 displays only the results without induced packet loss, chart 4.5 with 10% packet loss, and chart 4.6 with 20% packet loss. Chart 4.7 shows charts 4.4,4.5 and 4.6 together to provide a better visualization of how the protocols behave when there are different levels of packet loss.

With the line charts, we can observe that regardless of packet loss TCP protocols, when requiring more than one handshake, present a consistently higher RTT. This is better seen in the bar charts, where they remain constantly above the other protocols.

Another easily noticeable observation is that the CoAP protocol, when subjected to packet loss, suffers from high RTT values that far exceed those of the other protocols. The bar charts confirm this observation, showing CoAP's average RTT increasing significantly as induced packet loss rises.

Regarding the protocol under study, we can observe that QUIC's values remain close to those of TCP protocols when they perform a single handshake. However, this value diverges from TCP as more packet loss is introduced, sometimes reaching nearly double the values of TCP protocols. Nevertheless, these values continue to be significantly higher than those presented by CoAP and TCP protocols with more than one handshake.

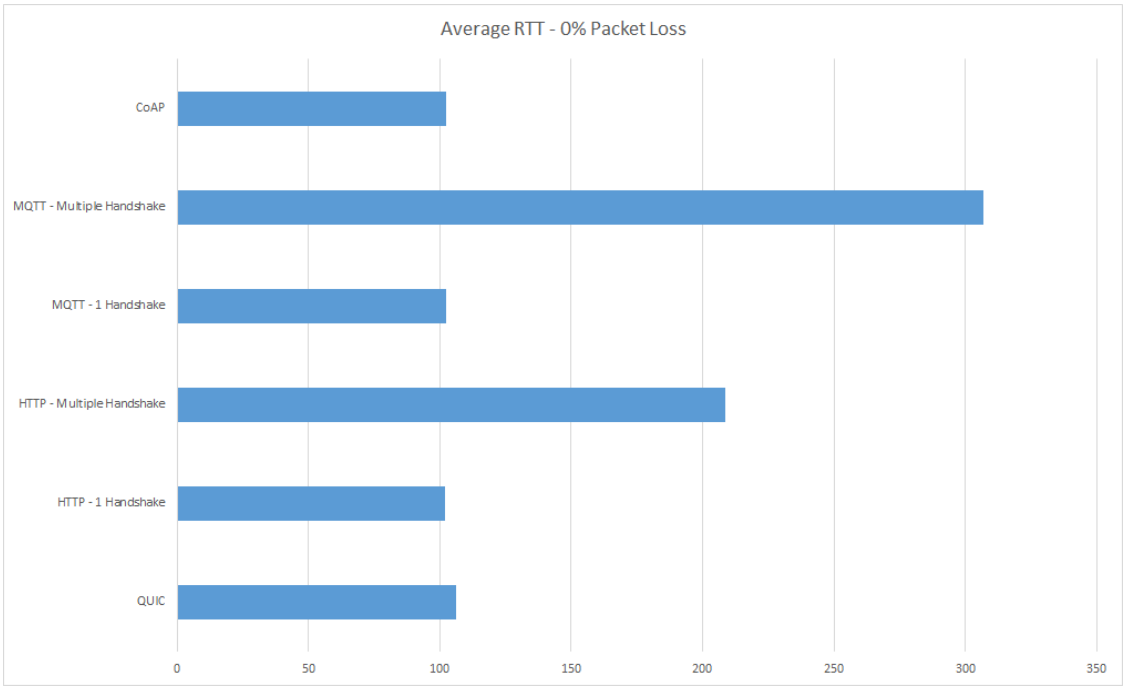

Figure 4.4: Average RTT - 0 %Packet Loss

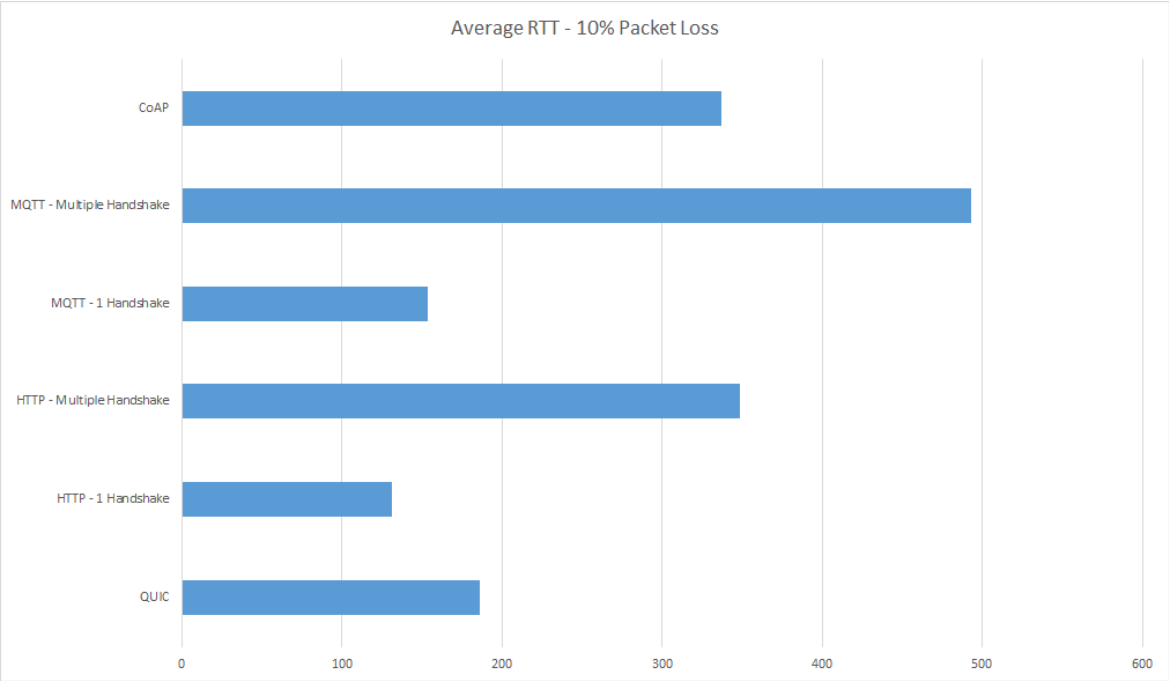

Figure 4.5: Average RTT - 10 %Packet Loss

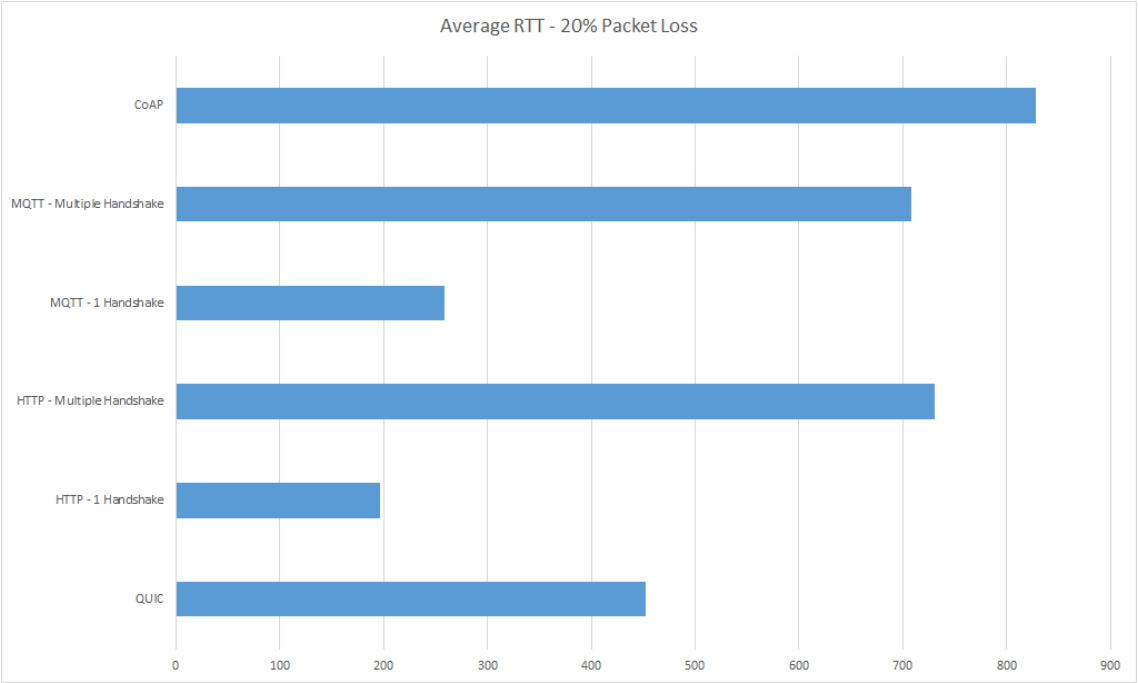

Figure 4.6: Average RTT - 20 %Packet Loss

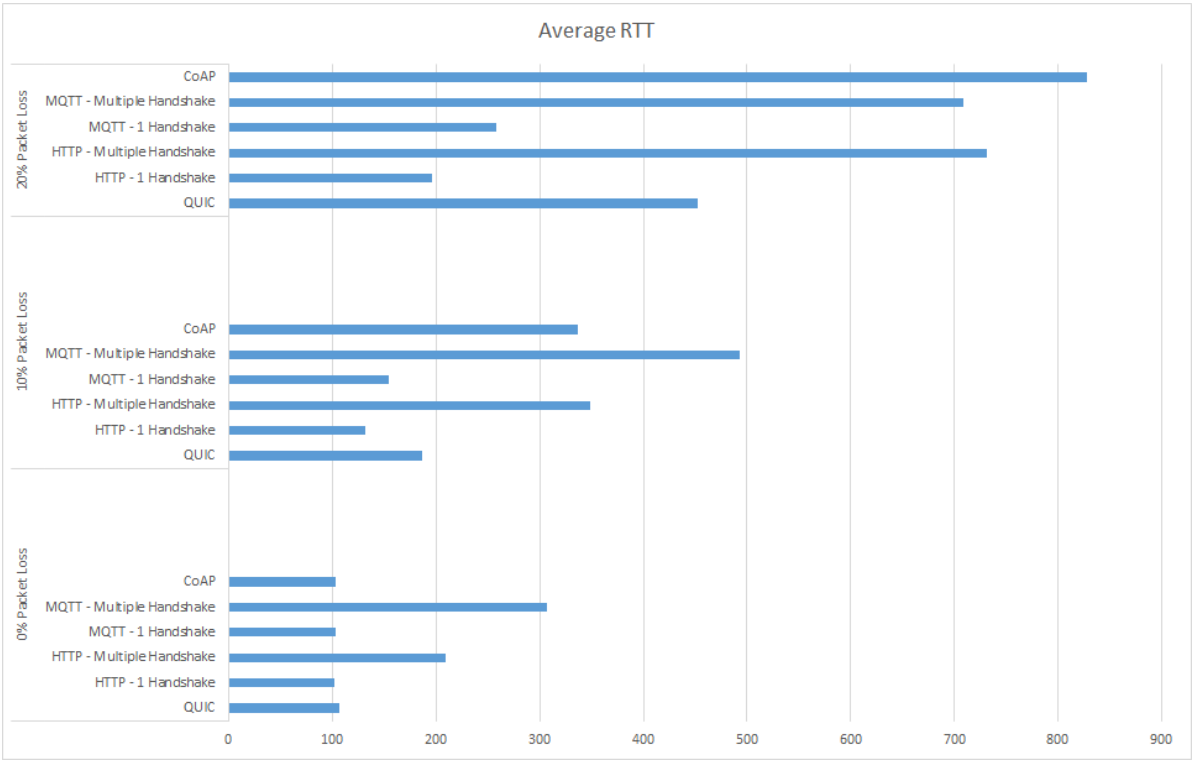

Figure 4.7: Average RTT

4.2 Discussion

In this section we discussed about the the results obtained in the implementation using the graphs made in the section before.

With the data obtained from the implementation, graphs were created to better interpret the results. Seven graphs were made, three of which are linear and show the RTT obtained for each transmitted message, demonstrating all the protocols being tested and their variants (TCP protocols with multiple handshakes). Each of these three graphs shows the same tests but with different induced packet loss percentages. The first has 4.1 no induced packet loss, while the subsequent ones have 10%4.2 and 20%4.3, respectively.

Three additional bar charts were produced to show the average RTT for each protocol and its variants, displaying all the tests side by side. A last bar chart was made to join the three bar charts in one so that we can better display the discrepancies between each level of induced packet loss.

RTT was used as the measurement value for the protocols' effectiveness due to its importance in communication and to demonstrate the effectiveness of existing protocol features, such as congestion control, packet retransmission, connection security, and support for network changes. Packet loss induction was used to demonstrate how the protocols behaved under degraded connection conditions and packet losses. This is particularly important when evaluating protocols that use UDP.

The test results were close to expectations but admirable given the limitations imposed during the development of this experiment. Due to the switch from native QUIC in Node.js to

the “node-quic” library, which uses GQUIC, a higher RTT was expected because it is older than QUIC and therefore less optimized and efficient in various aspects, such as the encryption that is integrated within it. However, we can observe that it achieved very good results when viewed in a general context. Starting by analyzing the graph where there is no packet loss, we can observe that all protocols obtained similar values, except for those using TCP, which constantly need to redo the handshake, as expected, showing values much higher than the other protocols. These results favor QUIC since, of all the protocols, this is the only one in which messages were encrypted due to having encryption built into it, whereas the other protocols would require external encryption, having to use, for example, the TLS protocol.
With the introduction of packet loss, we can observe that the results started to vary, with CoAP being the most affected. This protocol, when compared to the other protocols under normal usage, showed the highest average RTT. This is due to the use of UDP, which, as we already know, brings significant disadvantages in networks with high packet loss. In comparison, the protocols using TCP (MQTT and HTTP) continued to be extremely efficient when they did not need to constantly perform a handshake, which was also expected since TCP is a connection-oriented protocol. This means that it can maintain the integrity and correct sequence of transferred information, thus ensuring a reliable and fast connection. However, as in the tests conducted without induced packet loss, the HTTP and MQTT protocols did not perform well when they had to constantly redo the handshake, with MQTT showing the highest average RTT in this scenario among all protocols and situations tested. As for QUIC, we can observe that it remained successful, as even with an increase in packet loss, it maintained a very low RTT, demonstrating that despite the disadvantages of being a UDP-based protocol, it can overcome issues like those imposed in these situations.
Features such as improved congestion control, connection migration, 0-RTT, and multiplexing capability have really shown how advanced and reliable the protocol can be when used under these conditions. Even with packet loss increasing to 20%, this protocol continues to yield positive results.
On the other hand, the others, when faced with 20% packet loss, showed a trend similar to what was previously observed, with CoAP even deteriorating drastically as we can see in the bar graph. In the line graph, we can better observe the reason for this worsening of RTT, showing peaks that are on average twice as high as those presented by the other protocols.

4.3 Conclusion

With these results analyzed, we were able to obtain important information to understand the utility of QUIC in IoT. We observed that although QUIC achieves good results in situations where a good connection to the server is established, MQTT and HTTP can yield better results. The use of the publish/subscribe model and its simplicity in overhead allows MQTT to gain advantages in various IoT scenarios that QUIC cannot. However, when there is a need for high mobility of devices, high bandwidth, and short, intermittent connections, QUIC can surpass the other protocols. This is achieved through the use of multiplexing for data transmission using multiple streams over a single communication channel, connection

migration to keep devices with an active and functional connection, and 0-RTT for sending data quickly without the need for recent communication. These characteristics are ideal for the use of the QUIC protocol in autonomous or interconnected vehicles, remote medical devices, or portable personal health devices, as well as for mobile sensors like data-gathering drones.

However, despite showing advantages, the application of this protocol is quite limited due to a lack of research and incorporation into modern technologies. This has proven to be a major problem for the development of this study, as some of the technologies that would be suitable and relevant for investigating the QUIC protocol could not be applied. However, with the data we obtained, we can observe that even using an older version of QUIC (GQUIC) and not being able to use its native version in Node.js, the results achieved were remarkable.

Chapter 5

Conclusion

Based on the results obtained in this implementation, it was possible to prove the relevance and future capacity of the QUIC protocol when used in an IoT environment. Although the main objective of exploring this protocol using the RuraLTHINGS project for its application in a real situation has not been fully achieved, due to the current limitations it presents regarding its adoption in modern technologies. The use of the alternatives found demonstrated results more than sufficient for analysis and discussion. The development of the study in these technologies also showed the versatility of Node.js with its extensive community support and variety of libraries. Even more promising results are expected when the native integration of the QUIC protocol in Node.js is implemented.
The results obtained managed to demonstrate that even using a version of the studied protocol that is not as advanced as the current one (GQUIC), we were able to demonstrate good performance of the protocol while maintaining a low and stable RTT, especially in high mobility environments or unstable connections. However, outside of ideal implementation situations, protocols like MQTT still prove to be superior. This situation may change in the near future with the constant development and adaptation of the QUIC protocol in an IoT context.
With these points observed, we can conclude that QUIC is already ready to be integrated into IoT environments and should be considered as a viable option in most projects in this area. Its use can be particularly advantageous in autonomous vehicles, sensors, and surveillance systems that need to remain in constant motion (for example, drones) and in remote or personal health systems that require constant data transmission.
May this study shed light on the need for more implementations of the QUIC protocol, not only for a better understanding of it but also for better development of IoT technologies.

5.1 Future Work

Due to the conditions imposed that led to the use of GQUIC, the studies conducted may present outdated data regarding the functioning of the QUIC protocol today. Therefore, future studies are necessary to demonstrate the advantages that the QUIC protocol shows over this older version of it.
Studies that manage to utilize more commonly used protocols and technologies in IoT, in addition to those used in this study, can provide essential information for the better development of IoT applications. Furthermore, possible implementations of QUIC in cloud environments need to be developed, as currently, the opportunities to use this protocol in these situations are quite scarce.
Implementations of this protocol in already developed projects can be complex, but the advantages obtained from this may well justify the investment. Therefore, attempts to carry

out this type of implementation are advisable and can provide essential data for the development of QUIC. In the future, this study is also expected to be expanded with one of the initially planned objectives, which is the use of the RuraLTHINGS project.

Bibliography

[ort17] Introducing zero round trip time resumption (0-rtt) [online]. 2017. Available from: https://blog.cloudflare.com/introducing-0-rtt [cited 09 January 2024]. 8

[ort19] Even faster connection establishment with quic 0-rtt resumption [online]. 2019. Available from: https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/ [cited 09 January 2024]. 8

[ALA+24] George Amponis, Thomas Lagkas, Vasileios Argyriou, Panagiotis Radoglou-Grammatikis, Konstantinos Kyranou, Ioannis Makris, and Panagiotis Sarigiannidis. Channel-aware quic control for enhanced cam communications in c-v2x deployments over aerial base stations. *IEEE Transactions on Vehicular Technology*, 73(7):9320–9333, 2024. 11

[ALM22] Ahmed Alqattaa, Daniel Loebenberger, and Lukas Moeges. Analyzing the latency of quic over an iot gateway. In *2022 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, pages 1–6, 2022. 12, 14

[auv] What is quic? everything you need to know [online]. Available from: https://www.auvik.com/franklyit/blog/what-is-quic-protocol/ [cited 09 January 2024]. 4, 8

[BG15] Andrew Banks and Rahul Gupta. Mqtt version 3.1.1 plus errata 01. OASIS Standard Incorporating Approved Errata 01, 2015. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html. Available from: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html. 9

[cdn23] What is quic? how does it boost http/3? [online]. 2023. Available from: https://www.cdnetworks.com/media-delivery-blog/what-is-quic/ [cited 09 January 2024]. 8

[dht] Dht11–temperature and humidity sensor [online]. Available from: https://components101.com/sensors/dht11-temperature-sensor [cited 10 September 2024]. 19

[FBe20] How facebook is bringing quic to billions [online]. 2020. Available from: https://engineering.fb.com/2020/10/21/networking-traffic/how-facebook-is-bringing-quic-to-billions/ [cited 09 January 2024]. 4

[FZG+20] Fátima Fernández, Mihail Zverev, Pablo Garrido, José R. Juárez, Josu Bilbao, and Ramón Agüero. And quic meets iot: performance assessment of mqtt over quic. In *2020 16th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 1–6, 2020. 13, 14

[goo] Quic [online]. Available from: https://peering.google.com/#/learn-more/quic [cited 09 January 2024]. 8

[han] What happens in a tls handshake? [online]. Available from: https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/ [cited 09 January 2024]. 5

[Her21] Rolando Herrero. Analysis of quic transported coap. *SN Computer Science*, 2(2):62, 2021. Available from: https://doi.org/10.1007/s42979-021-00468-0. 13, 14

[IGK+23] Faheem Iqbal, Moneeb Gohar, Hanen Karamti, Walid Karamti, Seok-Joo Koh, and Jin-Ghoo Choi. Use of quic for amqp in iot networks. *Computer Networks*, 225:109640, 2023. Available from: https://www.sciencedirect.com/science/article/pii/S1389128623000853. 11, 12

[iot23] Number of iot devices (2023) [online]. 2023. Available from: https://explodingtopics.com/blog/number-of-iot-devices [cited 21 January 2024]. 9

[JFD+22] Sidna Jeddou, Fátima Fernández, Luis Diez, Amine Baina, Najid Abdallah, and Ramón Agüero. Delay and energy consumption of mqtt over quic: An empirical characterization using commercial-off-the-shelf devices. *Sensors*, 22(10), 2022. Available from: https://www.mdpi.com/1424-8220/22/10/3694. 8, 10

[JNCK23] Joong-Hwa Jung, Hye-Been Nam, Dong-Kyu Choi, and Seok-Joo Koh. Use of quic for coap transport in iot networks. *Internet of Things*, 24:100905, 2023. Available from: https://www.sciencedirect.com/science/article/pii/S2542660523002287. 9, 11, 12, 14

[LJBNR15] Robert Lychev, Samuel Jero, Alexandra Boldyreva, and Cristina Nita-Rotaru. How secure and quick is quic? provable security and performance analyses. In *2015 IEEE Symposium on Security and Privacy*, pages 214–231, 2015. 13

[MKM16] Péter Megyesi, Zsolt Krämer, and Sándor Molnár. How quick is quic? In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6, 2016. 13

[mosa] Eclipse mosquitto [online]. Available from: https://mosquitto.org/ [cited 10 September 2024]. 19

[mosb] Mosquitto : Mqtt [online]. Available from: https://medium.com/@bhagvankommadi/mosquitto-mqtt-2a352bd8f179 [cited 10 September 2024]. 19

[MQT22] Mqtt over quic: Next-generation iot standard protocol [online]. 2022. Available from: https://www.emqx.com/en/blog/mqtt-over-quic [cited 09 January 2024]. 10

[nod23a] Node.js: o que é, como funciona esse ambiente de execução javascript e um guia para iniciar [online]. 2023. Available from: https://www.alura.com.br/artigos/node-jssrsltid=AfmBOookgyHFBCTL1j_L1VHCclqgDR24jl3VuHDM_lBgIqbScqnxqcMg [cited 10 September 2024]. 18

[nod23b] O que é node.js? [online]. 2023. Available from: https://tecnoblog.net/responde/o-que-e-node-js-guia-para-iniciantes/ [cited 10 September 2024]. 18

[nod23c] Why node.js is single-threaded and how it scales [online]. 2023. Available from: https://medium.com/@codejuggler/why-node-js-is-single-threaded-and-how-it-scales-eb32f3fc4cbc [cited 10 September 2024]. 19

[oke] What is okeanos? [online]. Available from: https://okeanos.grnet.gr/support/faq/okeanos-what-is-okeanos/ [cited 10 September 2024]. 19

[PFR+17] Giorgos Papastergiou, Gorry Fairhurst, David Ros, Anna Brunstrom, Karl-Johan Grinnemo, Per Hurtig, Naeem Khademi, Michael Tüxen, Michael Welzl, Dragana Damjanovic, and Simone Mangiante. De-ossifying the internet transport layer: A survey and future perspectives. *IEEE Communications Surveys Tutorials*, 19(1):619–639, 2017. 3

[put23] What is putty? a comprehensive guide [online]. 2023. Available from: https://rushax.com/what-is-putty-a-comprehensive-guide/ [cited 10 September 2024]. 20

[rad23] Sistemas de proteção de gás radão [online]. 2023. Available from: https://www.soprema.pt/pt/gas-radao [cited 09 January 2024]. 16

[rasa] Raspberry pi 400 tech specs [online]. Available from: https://www.raspberrypi.com/products/raspberry-pi-400/specifications/ [cited 10 September 2024]. 20

[rasb] What is raspberry pi? models, features, and uses [online]. Available from: https://www.spiceworks.com/tech/networking/articles/what-is-raspberry-pi/ [cited 10 September 2024]. 19

[rfc] Quic: A udp-based multiplexed and secure transport [online]. Available from: https://datatracker.ietf.org/doc/html/rfc9000 [cited 10 September 2024]. 4

[roa18] The road to quic [online]. 2018. Available from: https://blog.cloudflare.com/the-road-to-quic [cited 09 January 2024]. 8

[rur] Ruralthings [online]. Available from: https://ruralthings.ubi.pt/ [cited 10 September 2024]. 15, 16

[Rur23] Projeto ruralthings arranca em parceria com os municípios do fundão e pinhel [online]. 2023. Available from: https://www.ubi.pt/Noticia/7655 [cited 09 January 2024]. 16

[SHB14] Zach Shelby, Klaus Hartke, and Carsten Bormann. The Constrained Application Protocol (CoAP). RFC 7252, June 2014. Available from: https://www.rfc-editor.org/info/rfc7252. 9

[SRS+23] Vidhya. S, Ramya. R, Selciya Selvan, Sabari Santhosh. S, Sabarishraj. K, and Swathi. R. A survey on modern innovative secured transport layer protocols on recent advances. In *2023 7th International Conference on Computing Methodologies and Communication (ICCMC)*, pages 1088–1093, 2023. 3

[tcp] Tcp/ip model [online]. Available from: https://www.geeksforgeeks.org/tcp-ip-model/ [cited 10 September 2024]. 3

[tlsa] Tls [online]. Available from: https://en.wikipedia.org/wiki/Transport_Layer_Security [cited 09 January 2024]. 5

[tlsb] What is tls [online]. Available from: https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/ [cited 09 January 2024]. 5

[wha23] What is coap? understanding the constrained application protocol [online]. 2023. Available from: https://www.radware.com/security/ddos-knowledge-center/ddospedia/coap/#HowDoesCoAPWork [cited 21 January 2024]. 11

[win] Winscp [online]. Available from: https://www.exavault.com/docs/winscp [cited 10 September 2024]. 20

[wir] Wireshark [online]. Available from: https://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html [cited 10 September 2024]. 20