Inatel

PhD Thesis

An Enhanced Multi-Protocol Middleware Solution for Internet of Things

Mauro Alexandre Amaro da Cruz

November/2021

An Enhanced Multi-Protocol Middleware Solution for Internet of Things

Mauro Alexandre Amaro da Cruz

Thesis presented to the Instituto Nacional de Telecomunicações (Inatel) – Brazil and Université Haute Alsace (UHA) - France as part of the requirements of an international joint supervision between University of Haute-Alsace (UHA) and National Institute of Telecommunications (Inatel), for obtaining the title of Doctor in Telecommunications (Inatel) and Doctor in Computer Science (UHA).

Supervisors

- Prof. Joel José Puga Coelho Rodrigues
- Prof. Pascal Lorenz
- Prof. Samuel Baraldi Mafra

Santa Rita do Sapucaí, Brazil - 2021

Cruz, Mauro Alexandre Amaro da
C957a
An enhanced multi-protocol middleware solution for internet of things/ Mauro Alexandre Amaro da Cruz. – Santa Rita do Sapucaí, 2021.
154 p.

Orientadores: Prof. Dr. Samuel Baraldi Mafra / Prof. Dr. Joel José Puga Coelho Rodrigues / Prof. Dr. Pascal Lorenz.
Tese de Doutorado em Telecomunicações – Instituto Nacional de Telecomunicações – INATEL.
Inclui bibliografia.

1. Internet das coisas 2. IoT 3. Middleware 4. Plataforma. 5. Modelo de Referência. 6. Doutorado em Telecomunicações. I. Mafra, Samuel Baraldi. II. Rodrigues, Joel José Puga Coelho. III. Lorens, Pascal. IV. Instituto Nacional de Telecomunicações – INATEL. V. Título.

CDU 621.39

Biblioteca "Ministro Olavo Bilac Pinto"

Ficha Catalográfica elaborada pelo Bibliotecário da Instituição
CRB6-2718 – Paulo Eduardo de Almeida

An Enhanced Multi-Protocol Middleware Platform for Internet of Things

Thesis presented to the Instituto Nacional de Telecomunicações (Inatel) – Brazil and Université Haute Alsace (UHA) – France as part of the requirements of an international joint supervision between University of Haute-Alsace (UHA) and National Institute of Telecommunications (Inatel), for obtaining the title of Doctor in Telecommunications (Inatel) and Doctor in Computer Science (UHA).

Approved on______/______/______ by the judging commission:

Prof. Joel José Puga Coelho Rodrigues
Supervisor – UFPI

Prof. Pascal Lorenz
Supervisor – UHA

Prof. Samuel Baraldi Mafra
Supervisor – Inatel

Prof. Jalel Bem-Othman
University of Paris

Prof. Plácido Rogério Pinheiro
UNIFOR

Prof. Victor Hugo C. Albuquerque
UFC

Prof. António Marcos Alberti
Inatel

Prof. José Marcos Câmara Brito

Head of PhD Course (Inatel)

Santa Rita do Sapucaí-MG – Brasil 2021

"I see now that the circumstances of one's birth are irrelevant. It is what you do with the gift of life that determines who you are"

Mewtwo – *Pokemon: The First Movie- Mewtwo Strikes Back (1998).*

Dedication

To my beloved wife, Sheila Cássia da Silva Janota, for her patience and support during this journey, and to my mother, Filomena Eugênia Passos Amaro, the bravest person I know. You always dreamed of a Doctor son who could cure people with his wisdom; this is the closest I could reach.

ACKNOWLEDGMENTS

First, I want to thank God for the gift of life.

A special thanks to the Ministry of Telecommunications, Information Technologies of Angola and Social Communication for conceding a scholarship. In particular, to the former Minister of Telecommunications, Information Technologies of Angola and Social Communication, José Carvalho da Rocha, as well as to the Director of the "*Fundo de apoio às Comunicações*", Américo dos Santos.

Thanks to my supervisor Prof. Joel Rodrigues, for his patience and friendship, who often came to rescue when the burnout was taking over. I will always hold your words of encouragement deeply in my heart. Thanks for the knowledge, dialogues, trust, and guidance. I will never forget our first talk "*Mauro, conhecemos-nos tarde nessa tua trajectória do Mestrado, mas creio que ainda vamos a tempo de fazer muitas coisas boas juntos*".

This work is partially funded by FCT/MCTES through national funds and when applicable co-funded EU funds under the Project UIDB/50008/2020.

Thank my supervisors Prof. Samuel Baraldi Mafra and Pascal Lorenz who was always very patient, using methods so that I could learn. Thanks for the advice, teachings, confidence and valuable guidance.

Thanks to Gisele Moreira dos Santos, from the Inatel academic records service. I said it many times and will keep reaffirming. You are the most important person at Inatel.

Thanks to the National Institute of Telecommunications - Inatel, Brazil and Université Haute Alsace – UHA, France, for its support to this work, as well as all the Professors for their precious teachings, especially Professors José Marcos Câmara Brito, Antônio Marcos Alberti, Luciano Leonel Mendes, and Felipe Figueiredo. I also thank Leonardo Maia, coordinator of Inatel's international office.

I thank my friends and colleagues Elvira Diogo, Luis Affonso, José Vitor, Kellow Pardini, Yara Mendes, Flávia Larisse, Vitor Figueiredo, Eduardo Teixeira, Diego Zúñiga, Heitor de Paula, and Bruno Caputo. Also, a shoutout to Lucas Abbade, for the productive discussions and cooperation in and out of research, we made a great team.

I thank Márcia Abbade, for helping me during a turbulent time of my life, I have no words to express how thankful I am.

I thank my family, brothers, uncles, and nephews who were always there for me with the advices and moral support. A special thanks to my uncle Dino and my late aunt Melita, because without their support, I could not obtain the Master or the PhD degree.

I thank my dear Father Lindo and my dear Mother Mena, I love you both very much.

A special thanks to my wife, Sheila Janota, for her love, support, understanding, kindness, and just being my better half .

Abstract

In Internet of Things (IoT), data is handled and stored by software known as middleware (located on a server). The evolution of the IoT concept led to the construction of many IoT middleware, software that plays a key role since it supports the communication among devices, users, and applications. Several aspects can impact the performance of a middleware. Based in a deep review of the related literature and in the proposal of a Reference Model for IoT middleware, this thesis proposes a new IoT middleware, called In.IoT, a scalable, secure, and innovative middleware solution based on a deep review of the state of the art and following the reference middleware architecture that was proposed along with this research work. In.IoT addresses the middleware concerns of the most popular solutions (security, usability, and performance) that were evaluated, demonstrated, and validated along this study, and it is ready and available for use. In.IoT architectural recommendations and requirements are detailed and can be replicated by new and available solutions. It supports the most popular application-layer protocols (MQTT, CoAP, and HTTP). Its performance is evaluated in comparison with the most promising solutions available in the literature and the results obtained by the proposed solution are extremely promising. Furthermore, this thesis studies the impact of the underlying programming language in the solution's overall performance through a performance evaluation study that included Java, Python, and Javascript, identifying that globally, Java demonstrates to be the most robust choice for IoT middleware. IoT devices communicate with the middleware through an application layer protocol that may differ from those supported by the middleware, especially when it is considered that households will have various devices from different brands. The thesis offers an alternative for such cases, proposing an application layer gateway, called MiddleBridge. MiddleBridge translates CoAP, MQTT, DDS, and Websockets messages into HTTP (HTTP is supported by most IoT middleware). With MiddleBridge, devices can send a smaller message to an intermediary (MiddleBridge), which restructures it and forwards it to a middleware, reducing the time that a device spends transmitting. The proposed solutions were evaluated in comparison with other similar solutions available in the literature, considering the metrics related to packet size, response times, requests per second, and error percentage, demonstrating their better results and tremendous potential. Furthermore, the study used XGBoost (a machine learning technique) to detect the occurrence of replication attacks where an attacker obtains device credentials, using it to generate false data and disturb the IoT environment. The obtained results are extremely promising. Thus, it is concluded that the proposed approach contributes towards the state of the art of IoT middleware solutions.

Keywords

Internet of Things; IoT; IoT Architecture; Middleware; Reference Model; Platform; CoAP; MQTT; HTTP; In.IoT; Performance Evaluation; Bridge; Gateway; Application.

Resumo

Na Internet das Coisas (IoT), os dados são tratados e armazenados por um software conhecido como middleware (localizado em um servidor). A evolução do conceito de IoT levou à construção de muitos middlewares de IoT, um software que desempenha um papel fundamental, pois suporta a comunicação entre dispositivos, usuários e aplicações. Nesse sentido, a tese propõe um novo middleware IoT, denominado In.IoT, uma solução de middleware escalável, segura e inovadora baseada em uma profunda revisão do estado da arte e seguindo a arquitetura de referência de middleware que foi proposta junto com este trabalho de pesquisa. O In.IoT endereça as preocupações das soluções de middleware mais populares (segurança, usabilidade e desempenho) que foram avaliadas, demonstradas e validadas ao longo deste estudo e está pronto e disponível para uso. As recomendações e requisitos da arquitetura do In.IoT são detalhados e podem ser replicados por soluções novas e também pelas já existentes. O In.IoT suporta os protocolos de camada de aplicação mais populares (MQTT, CoAP e HTTP). Seu desempenho é avaliado e comparado com as soluções mais promissoras disponíveis na literatura e os resultados obtidos pela solução proposta são extremamente promissores. Além disso, a tese estuda o impacto da linguagem de programação usada na construção do software no desempenho geral da solução por meio de um estudo de avaliação de desempenho que incluiu Java, Python e Javascript, identificando que, globalmente, Java demonstra ser a escolha mais robusta para middleware IoT. Os dispositivos IoT comunicam-se com o middleware por meio de um protocolo de camada de aplicação que pode diferir daqueles suportados pelo middleware, especialmente quando se considera que as residências terão vários dispositivos de diferentes marcas. A tese oferece uma alternativa para tais casos, propondo um gateway da camada de aplicação denominado MiddleBridge. MiddleBridge converte mensagens CoAP, MQTT, DDS e Websockets em HTTP (HTTP é suportado pela maioria dos middleware). Com o MiddleBridge, os dispositivos podem enviar uma mensagem menor para um intermediário (MiddleBridge), que a reestrutura e encaminha para um middleware, reduzindo o tempo que um dispositivo gasta transmitindo. As soluções propostas foram comparadas com outras soluções semelhantes disponíveis na literatura, considerando as métricas relacionadas ao tamanho do pacote, tempos de resposta, solicitações por segundo e percentagem de erro, demonstrando enome potencial. Além disso, o estudo usou o XGBoost (uma técnica de ML) para detectar a ocorrência de ataques de replicação onde um invasor obtém credenciais do dispositivo, usando-as para perturbar o ambiente IoT. Os resultados foram extremamente promissores. Assim, conclui-se que as melhorias propostas contribuem para o estado da arte das soluções de middleware IoT.

Palavras chave

Internet das Coisas; IoT; Arquitectura IoT; Middleware; Plataforma; Modelo de Referência; CoAP; MQTT; HTTP; In.IoT; Avaliação de Desempenho; Bridge; Gateway.

Contents

Abstract xiii
Keywords xiv
Resumo xv
Palavras chave xvi
Contents xvii
List of Figures xxi
List of Tables xxiii
List of Algorithms xxv
List of Abbreviations and Acronyms xxvii
1 Introduction 1
1.1 Motivation 2
1.2 Problem definition 2
1.3 Research objectives 3
1.4 Main contributions 4
1.5 Publications 6
1.6 Thesis statement 9
1.7 Document organization 9
2 Related Work 11
2.1 Device classification regarding processing capabilities 12
2.2 Classification regarding communication capabilities 13
2.3 IoT device classification regarding processing and connectivity capabilities 13
2.3.1 Architectural reference model for LPLC devices 14
2.3.2 Architectural reference model for LPHC devices 15
2.3.3 Architectural reference model for HPHC devices 16
2.4 Connecting objects to the Internet using IoT 17
2.5 Message exchange patterns 18
2.6 IoT Application layer protocols 19

2.7 A global IoT standard ... 21

2.8 IoT Middleware ... 23

2.9 IoT middleware reference architecture ... 26

2.10 Authorization mechanisms for IoT middleware ... 29

2.11 Main issues of existing solutions ... 31

2.12 Summary ... 33

3 Performance Comparison of IoT Middleware Solutions ... 35

3.1 Qualitative metrics for IoT middleware ... 35

3.2 Quantitative metrics for IoT middleware ... 36

3.3 Performance evaluation through multi-criteria decision making ... 38

3.3.1 Preference Ranking Organization Method for Enrichment Evaluation (PROMETHEE) ... 38

3.3.2 Performance evaluation through PROMETHEE ... 39

3.4 Summary ... 44

4 Bridging Application Layer Protocols to HTTP ... 45

4.1 Application layer gateways ... 46

4.2 Bridging from IoT Protocols to HTTP ... 48

4.3 Orion context broker use case ... 52

4.4 Packet Size ... 54

4.5 Response times ... 55

4.6 Summary ... 56

5 Performance Comparison of Programming Languages for Internet of Things Middleware ... 57

5.1 Previous performance evaluation studies regarding programming languages ... 58

5.2 Programming languages for REST APIs ... 60

5.3 Experimentation scenario ... 62

5.4 Percentage of failed requests ... 65

5.5 Requests per second ... 66

5.6 Summary ... 69

6 In.IoT – A New Middleware for Internet of Things ... 71

6.1 Microservices overview 71
6.2 The In.IoT architecture 72
6.2.1 Data storage 72
6.2.2 Microservices and backward compatibility 73
6.2.3 Security 74
6.2.4 Improving overall performance 76
6.3 Construction of the In.IoT middleware 78
6.3.1 In.IoT features relative to the reference modules 79
6.4 Performance evaluation and demonstration 83
6.4.1 Packet size to publish data 84
6.4.2 Response times 85
6.4.3 Real-life deployment and usage 87
6.5 Summary 88
7 OLP – A RESTful Open Low-Code Platform 89
7.1 No-Code, Low-Code, and Traditional Approaches 90
7.1.1 Non-Functional Requirements 94
7.1.2 Functional Requirements 96
7.1.3 Architecture 98
7.2 OLP Development and Demonstration 100
7.2.1 Demonstration 103
7.3 Summary 104
8 Detecting Compromised IoT Devices Through XGBoost 107
8.1 Security threats in IoT environments and similar studies 108
8.1.1 Security threats for IoT platforms and networks 108
8.1.2 Security Threats for IoT Devices 110
8.1.3 Similar studies 111
8.2 XGBoost 113
8.3 Experimentation scenario and result analysis 116
8.3.1 Result analysis 118

8.4 Summary 120

9 Conclusion and Future Works 123

9.1 Learned lessons 124

9.2 Final remarks 125

9.3 Future works 129

References 131

List of Figures

Figure 1 –*The IoT landscape across various verticals.* 11
Figure 2 – *Architectural reference model for Low Power Low Connectivity (LPLC) devices.* 14
Figure 3 – *Architectural reference model for Low Power High Connectivity (LPHC) devices.* 15
Figure 4 – *Architectural reference model for High Power High Connectivity (HPHC) devices.* 16
Figure 5 – *Illustration of the communication (a) without middleware and (b) with middleware.* 24
Figure 6 – *IoT middleware reference model.* 27
Figure 7 – *Results of the qualitative comparison considering application-layer protocols, security features, and number of releases.* 41
Figure 8 – *Results of the quantitative comparison considering response time, packet size, and error percentage.* 42
Figure 9 – *Results of the global comparison summarized for the five studied scenarios* 43
Figure 10 – *Illustration of a bridge operation where a gateway receives messages through protocol A and translates them to a protocol B.* 47
Figure 11 – *MiddleBridge's graphical user interface is configured to send data to the Orion context broker through the MQTT protocol.* 49
Figure 12 – *Flowchart explaining the MiddleBridge algorithm.* 51
Figure 13 – *Illustration of a bracelet (i.e., an object) sending a small message to MiddleBridge that is reconstructed and sent to the IoT platform.* 52
Figure 14 – *Photo of the hardware used in the experiments: a) Dell Precision 5820 that hosts Orion Context Broker b) Raspberry Pi 3 that hosts MiddleBridge; c) Dell Precision 5820 that hosts Apache Jmeter, used to generate the CoAP, MQTT, Websockets, and direct HTTP requests.* 53
Figure 15 – *Analysis of the packet size (in bytes) of a single request with MQTT, CoAP, Websockets, and a direct HTTP Request to Orion Context Broker.* 54
Figure 16 – *Analysis of the average response time (in milliseconds) with MQTT, CoAP, and WebSockets, and direct HTTP communication with a server.* 55
Figure 17 – *Most popular programming languages on Stack Overflow, 2019.* 61
Figure 18 – *Most popular web frameworks on Stack Overflow, 2019.* 62
Figure 19 – *Data Insertion flowchart used in the experimentation scenario.* 63
Figure 20 – *Percentage of failed requests by number of parameters.* 66
Figure 21 – *Successful requests per second for 1 parameter.* 67
Figure 22 – *Successful requests per second for 10 parameters.* 67
Figure 23 – *Successful requests per second for 100 parameters.* 68
Figure 24 – *Successful requests per second for 1000 parameters.* 68
Figure 25 – *Illustration of In.IoT microservices architecture and data storage.* 74
Figure 26 – *Response time analysis for 5 000 concurrent users where 1 parameter were sent considering In.IoT and the modifications presented in Expressions (1), (2), and (3).* 78
Figure 27 – *Example of (a) an auto-register JSON from a coffee machine and (b) the unique credentials returned by In.IoT* 80
Figure 28 – *Example of how the auto-registration function can be used by a device with three operation modes: i) receiver, ii) hotspot, and iii) configured.* 82
Figure 29 – *Packet size analysis of a single successful request where 1, 15, and 100 parameters were sent considering In.IoT, Linksmart, Konker, Orion, and Sitewhere middleware.* 84
Figure 30 – *Response time analysis for 5 000 concurrent users where 1, 15, and 100 parameters were sent considering In.IoT, Linksmart, Konker, and Orion middleware* 86

Figure 31 – *Response time analysis for 10 000 concurrent users where 1, 15, and 100 parameters were sent considering In.IoT, Linksmart, and Orion middleware.* 86
Figure 32 – *In.IoT User Interface that was used for a Smart energy Meter.* 87
Figure 33 – *The general architecture of low-code platforms.* 99
Figure 34 – *Pipeline detailing how the code is transformed from the visual representations into a fully-fledged application.* 101
Figure 35 – *Interface creation screen for an application that pulls the temperature from a website and displays its value on a canvas.* 103
Figure 36 – *Logic screen for an application that pulls the temperature from a website and displays its value on a canvas.* 104
Figure 37 – *The main function of the source code generated by the platform from the steps presented in Figures 35 and 36.* 104
Figure 38 – *Illustration of a decision tree that tries to determine the probability of a state voting for a yellow (represented as False and located on the left side) or a purple party (represented as True and located on the right side).* 114
Figure 39 – *Confusion matrix results of the test data – 1 means attack.* 119
Figure 40 – *Feature importance determined by the trained model.* 120

List of Tables

Table 1 – *Summarized comparison among Websockets, DDS, CoAP, and MQTT protocols.* 21
Table 2 – *Qualitative Metrics Summarized.* 40
Table 3 – *Quantitative Metrics Summarized.* 41
Table 4 – *No/Low-Code main characteristics.* 93
Table 5 – Grid search parameters to find the optimal training parameters considering η, γ, λ, depth, scale, rounding, accuracy, and f1 score. 117

List of Algorithms

Algorithm 1 – PROMETHEE II algorithm 39

List of Abbreviations and Acronyms

IoT – Internet of Things

IIoT – Industrial Internet of Things

OASIS – Organization for the Advancement of Structured Information Standards

RPS – Requests Per Second

MQTT – Message Queuing Telemetry Transport

CoAP – Constrained Application Protocol

DDS – Data Distribution Service

RR – Request-Response

PubSub – Publish/Subscribe

HTTP – Hypertext Transfer Protocol

TCP – Transmission Control Protocol

REST – Representational State Transfer

UDP – User Datagram Protocol

GUI – Graphical User Interface

API – Application Programming Interface

JSON – Javascript Object Notation

SDK – Software Development Kit

XML – eXtensible Markup Language

SQL – Structured Query Language

NoSQL – Not only SQL

AI – Artificial Intelligence

MAC – media access control address

IP – Internet Protocol

PaaS – Platform as a Service

MCDM – multiple-criteria Decision Making

PROMETHEE – Preference Ranking Organization Method for Enrichment Evaluation

XMPP – Extensible Messaging and Presence Protocol

OMG – object management group

RFC – Request for Comment

IETF – Internet Engineering Task Force

ALB – Application-Layer Bridge

ALG – Application-Layer Gateway

GNU – GNU's Not Unix

GPL – GNU General Public License

LAN – Local Area network

ACK – Acknowledgement

NACK – Negative Acknowledgment

JVM – Java Virtual Machine

CPU – Central Processing Unit

RAM – Random Access Memory

JWT – Json Web Token

SSD – Solid-State Drive

AWS – Amazon Web Services

EKS – Elastic Kubernetes Service

1 Introduction

Internet of Things (IoT) is a paradigm that intends to connect every object (the "things") to the Internet [1][2]. These objects can belong to the physical world (living organisms and inanimate objects) or a virtual world (simulating or representing a real environment) [3]. The objects are connected to exchange and collect data, turning the surrounding environment smarter [4]. Humanity is far from connecting everything (every object). However, the milestone of 20 billion Internet-connected objects was likely breached in 2020 [5] and could reach 75 billion in 2030 [6]. Most IoT devices present reduced size and constrained resources (disk space, memory, processing power) [7]. Naturally, such a high number of devices raises various challenges related to their physical size and constrained resources [8][9]. These challenges include connectivity, communication, security, standardization, user trust, scalability, and much more [10][11].

IoT devices' few resources contrast with current Internet where clients may be desktops, laptops, and smartphones. In the current Internet, efficacy (producing the expected results) is prioritized over efficiency (how the results were achieved). Users barely notice this tradeoff because they use strong devices, such as laptops and smartphones. IoT favors efficiency because fewer resources are available. In practice, if a developer writes an application for a smartphone with poor code, at most, there will be performance issues that can later be solved. IoT developers cannot take the same approach because a poor-written code might not even run on the device.

The main difference between a constrained IoT device and a smartphone (which is also considered an IoT device) lies in the fact that an IoT device's primary concern is to consume the least possible resources (i.e., energy and memory) without compromising its objectives. Some literature, even suggests using smartphones as mobile gateways for IoT devices [12][13][14]. Despite being constrained in terms of resources (by current standards), IoT devices are smaller, in size, and more powerful

than most early computers. The entire IoT vision revolves around data. Therefore, an IoT middleware is present in every solution because it is the software responsible for collecting such data [15][16].

1.1 Motivation

The IoT concept of connecting regular objects to the Internet is changing the way technology is perceived and experienced in daily life [17]. These objects are rapidly growing in numbers and collectively generate a respectable amount of data, which is sent to a software named IoT middleware. The IoT middleware not only receives the data from devices but is also responsible for enabling communication among devices, users, and applications [18]. This software is popular in IoT implementations because it accelerates the development and deployment through the various functionalities it provides [19]. Since most of the offered functionalities are generic, most IoT middleware solutions need auxiliary applications to achieve very specific goals. With such an important role, it is no surprise that various solutions and studies were published on IoT middleware topics, and selecting an IoT middleware for a solution is crucial for a successful IoT scenario [20].

Despite having such an important role, most IoT middleware solutions possess various usability issues, their performance could be optimized, and security vulnerabilities are not uncommon [21]. In this sense, this thesis will focus on developing a novel middleware solution capable of addressing these issues and offer new contributions in the topic.

1.2 Problem definition

With the course of technological evolution, Internet of Things (IoT) has become one of the most significant current trends in terms of technology. It considers entities are interconnecting from the digital, virtual, and physical world, such as objects and appliances, to the Internet, with minimal human interference [22]. This technology allows users to control their smart objects and visualize data from sensors through the Internet. IoT is also attractive for industry, through a concept that is known as Industrial IoT (IIoT) [23][24]. In general, IoT optimizes remote monitoring and control with less

costs, which is crucial for the industry. Furthermore, the data produced by smart appliances can reveal users' habits and trends, that can be leveraged to improve several services [25]. IIoT can also reduce CAPEX (Capital Expenditures) and OPEX (Operation Expenses) [26].

Significant growth is estimated for the IoT market, given its impact in various areas, such as healthcare, energy, cities, homes, and agriculture [27]. It is estimated that IoT related spending could reach $1 trillion by the year 2022 [28]. Therefore, IoT will be a lucrative industry and improve its users' quality of life, which means that the IoT concept can be applied to nearly every environment such as transportation [29], energy grids [30], industry [31], agriculture [32], healthcare [33], smart homes [34], smart cities [35], and many more.

Most IoT solutions do not offer an easy way to integrate new devices into the environment without compromising security. This is mainly because IoT middleware solutions do not support it. Thus, integrating new devices is a manual and time-consuming task that is difficult to accomplish and is counter-intuitive to the IoT concept of "minimal human interference." The existing solutions rarely address critical security considerations, such as individual device credentials and the distinction between admin users (human users) and devices. Also, there are few studies that evaluate the impact of the underlying programming language in the overall performance of middleware solutions. Another issue is that there is no universal standard to connect devices, which means that sometimes a device can only communicate in a protocol not supported by the IoT middleware. Finally, most of the available solutions and studies neglect critical security and privacy issues related to the increasingly popular MQTT protocol.

1.3 Research objectives

The thesis' main objective is to propose an open-source solution with multi-protocol support that outperforms the existing solutions regarding response times and security. Furthermore, the proposed solution should be easily scalable when multiple concurrent users are present (the available literature demonstrates that 5000 concurrent users sending 15 variables is a good starting point to demonstrate scalability) [5]. To reach this global objective, the following partial objectives were defined:

- Perform a comprehensive literature review regarding IoT middleware solutions, identifying their main characteristics, limitations, performance evaluation mechanisms, and performance evaluation metrics;
- Review the most relevant application layer protocols for IoT, propose a mechanism that allows devices that support protocols that are incompatible with the middleware to still communicate with the middleware and evaluate its performance;
- Study the most popular programming languages and evaluate their effect on the performance of middleware solutions and propose a novel IoT middleware able to surpass the existing solutions regarding performance and better attend to the IoT application requirements using the identified programming language;
- Select the most relevant open-source middleware solutions for a comparison study with the proposed solution and evaluate the proposed middleware solution's performance, considering the identified requirements and defined metrics and compare its performance against the identified relevant solutions;
- Study and propose a novel solution that facilitates the development of new applications using the “low-code” concept;
- Study and propose mechanisms that detects the occurrence of replication attacks where an attacker obtains device credentials and impersonates a device.

1.4 Main contributions

The first contribution of this thesis is a deep review of the state of the art in Middleware solutions for IoT, the classification of IoT devices according to the processing power and communication capabilities. Next, a reference architecture for IoT middleware is proposed, and the main issues with the most relevant open-source middleware solutions is presented. This study is presented in Chapter 2. The classification for IoT devices was published on the **ITU-T Y.4460 - Architectural Reference Model of Devices for IoT Applications** [36]. The reference architecture was published in the Journal of **IEEE Internet of Things Journal**, ISSN: 2327-4662, Volume 5, no. 2, April 2018, pp. 871-883, DOI: 10.1109/JIOT.2018.2796561 [15].

The second contribution is a review of the most popular qualitative and quantitative evaluation metrics for IoT middleware that were used across various studies to compare middleware solutions. The traditional comparison is only capable of

determining the best solution for metric and this is reflected in the existing performance evaluation studies. Then, the usage of Multi-criteria decision-making techniques and how they can be used to combine various performance metrics (either qualitative or quantitative) in the same comparison study and identify the best solution according to a chosen scenario. This study is presented in Chapter 3. A performance evaluation study that included eleven (11) middleware solutions was published on the **Journal of Network and Computer Applications**, ISSN: 1084-8045, Volume 109, May 2018, pp. 53-65, DOI: 10.1016/j.jnca.2018.02.013 [16]. The MCDM study was published on **2018 IEEE Global Communications Conference (GLOBECOM),** Abu Dhabi, United Arab Emirates, December 9-13, DOI: 10.1109/GLOCOM.2018.8647381 [37].

The third contribution is the proposal of MiddleBridge, a simple and efficient solution for devices that are incompatible with a chosen middleware because of the lack a certain application-layer protocol. MiddlBridge supports MQTT, CoAP, Websockets, and DDS protocols. The study is presented in Chapter 4 and was published in the Journal of **Future Generation Computer Systems (FGCS)**, ISSN: 0167739X, Volume 97, August 2019, pp. 145-152, DOI: 10.1016/j.future.2019.02.009 [38].

The fourth contribution consists on analyzing the influence of the programming language in the solutions' performance. The study reviewed the most popular programming languages for building IoT middleware and determined the best programming language for this task. The study is presented in Chapter 5 and was published in the Journal of **Transactions on Emerging Telecommunications Technologies**, ISSN: 2161-3915, Volume 31, no. 12, December 2020, e3891, DOI: 10.1002/ett.3891 [39].

The fifth contribution is the proposal of In.IoT, a middleware solution that is based on a reference architecture for IoT middleware. In.IoT surpasses the performance of the existing solutions under heavy load, while also providing innovations regarding security and usability. The study is presented in Chapter 6 and was accepted for publishing in the **IEEE Internet of Things Journal**, ISSN: 2327-4662, DOI: 10.1109/JIOT.2020.3041699 [40].

The sixth contribution is the proposal of OLP, a low-code platform that transforms visual representations into functional software, allowing anyone to become a developer and create customs applications for In.IoT or other middleware solutions.

The study is presented in Chapter 7 and was accepted for publishing in the **Future Internet**, ISSN: 1999-5903, DOI: 10.3390/fi13100249.

The seventh contribution is the proposal of a security mechanism based on machine learning that improves the security in IoT environments through the detection of the replication attacks where an attacker obtains device credentials causing disruptions in the environment. It uses XGBoost, a machine learning technique to detect the attackers, the work is presented in Chapter 8 and was submitted to an International journal.

1.5 Publications

During this research, six (6) scientific papers that are directly related to the research topic were published and one (1) was submitted and is awaiting review. Four (4) scientific papers were published in International Journals, one (1) was submitted to an International Journal and is awaiting the review notification, and two (2) on International conferences. Furthermore, an ITU-T Recommendation was approved, a software was registered, and four (4) scientific papers that are not directly related to the core of the thesis were published (one journal and three conference papers). Other two (2) studies were published in International Journals before the beginning of the thesis but are directly related to this thesis in the sense that their conclusions and future works set the ground rules for the current study.

Publication in International Journals

1. **Mauro A. A. da Cruz**, Joel J. P. C. Rodrigues, Pascal Lorenz, Petar Solic, Jalal Al-Muhtadi, Victor H. C. de Albuquerque, "A proposal for bridging application layer protocols to HTTP on IoT solutions," *Future Generation Computer Systems*, Elsevier, Vol. 97, August 2019, pp. 145–152, DOI: 10.1016/j.future.2019.02.009.

2. Lucas R. Abbade, **Mauro A. A. da Cruz**, Joel J. P. C. Rodrigues, Pascal Lorenz, R. A. L. Rabelo, Jalal Al-Muhtadi, "Performance comparison of programming languages for Internet of Things middleware," *Transactions on Emerging*

Telecommunications Technologies, Wiley, Vol. 31, No. 12, December 2020, Paper Id: e3891, DOI: 10.1002/ett.3891.

3. **Mauro A. A. da Cruz**, Joel J. P. C. Rodrigues, Pascal Lorenz, Valery Korotaev, Victor H. C. de Albuquerque, "In.IoT – A new Middleware for Internet of Things," *IEEE Internet of Things Journal*, vol. 8, no. 10, May 2021, pp. 7902-7911, DOI: 10.1109/JIOT.2020.3041699.

4. **Mauro A. A. da Cruz**, Joel J. P. C. Rodrigues, Jalal Al-Muhtadi, Valery Korotaev, Victor H. C. de Albuquerque, "A reference model for Internet of Things middleware," *IEEE Internet of Things Journal*, Vol. 5, No. 2, April 2018, pp. 871–883, DOI: 10.1109/JIOT.2018.2796561.

5. **Mauro A. A. da Cruz**, Joel J. P. C. Rodrigues, Arun K. Sangaiah, Jalal Al-Muhtadi, Valery Korotaev, "Performance evaluation of IoT middleware," *Journal of Network and Computer Applications*, Vol. 109, May 2018, pp. 53–65, DOI: 10.1016/j.jnca.2018.02.013.

6. **Mauro A. A. da Cruz**, Heitor T. L. de Pauka, Bruno P. G. Caputo, Samuel B. Mafra, Pascal Lorenz, Joel J. P. C. Rodrigues, "OLP – A RESTful Open Low-code Platform," *Future Internet*, Vol. 13, No. 10, Sep. 2021, p. 249, DOI: 10.3390/fi13100249.

7. **Mauro A. A. da Cruz**, Lucas R. Abbade, Pascal Lorenz, Samuel B. Mafra, Joel J. P. C. Rodrigues, "Detecting Compromised IoT Devices through XGBoost," *Submitted to International Journal*.

ITU-T Recommendation

1. Joel J. P. C. Rodrigues, **Mauro A. A. Da Cruz**, Tiago G. F. Barros; João A. M. Zanon; Rodrigo dos S. Santos; Daniel A. G. Costa, “ITU-T Y.4460 - Architectural Reference Model of Devices for IoT Applications,” Genebra,

Switzerland: International Telecommunications Union, 2019 (ITU-T Recommendation - International Standard).

Registered Software

1. **Mauro A. A. da Cruz,** Joel J. P. C. Rodrigues, “In.IoT, registry request of computer program in Brazil—RPC N BR 512018051862-1” Instituto Nacional de Telecomunicações, Santa Rita do Sapucaí, Brazil, Tech. Rep. 1, Oct. 2018.

Publication in International Conferences

1. **Mauro A. A da Cruz**, Joel J. P. C. Rodrigues, Ellen S. Paradello, Pascal Lorenz, Petar Solic, Victor H. C. Albuquerque, "A proposal for bridging the message queuing telemetry transport protocol to HTTP on IoT solutions," in *3rd International Conference on Smart and Sustainable Technologies (Splitech 2018)*, Split, Croatia, June 26-29, 2018, pp. 1-5.

2. **Mauro A. A da Cruz**, Guilherme A. B. Marcondes, Joel J. P. C. Rodrigues, Pascal Lorenz, Plácido R. Pinheiro, "Performance Evaluation of IoT Middleware through Multicriteria Decision-Making," in *IEEE Global Communications Conference (IEEE GLOBECOM 2018), Abu Dhabi, United Arab Emirates*, December 09-13, 2018, pp. 1-5, DOI: 10.1109/GLOCOM.2018.8647381.

Other Publications

During this PhD, four contributions that are not directly related to the core research were published. The first was in a reputed International Journal and the other three were on International conferences, as follows:

1. Ivo B. F. de Almeida, Luciano L. Mendes, Joel J. P. C. Rodrigues, **Mauro A. A. da Cruz**, "5G Waveforms for IoT Applications," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3 pp. 2554–2567, 3rd Quarter 2019, DOI: 10.1109/COMST.2019.2910817.
2. Rodolfo R. Rodrigues, Joel J. P. C. Rodrigues, **Mauro A. A. da Cruz**, Ashish Khanna, Deepak Gupta, "An IoT-based automated shower system for smart

homes," in *International Conference on Advances in Computing, Communications and Informatics (ICACCI 2018)*, Bangalore, India, September 19-22, 2018, pp. 254-258, DOI: 10.1109/ICACCI.2018.8554793.

3. **Mauro A. A da Cruz**, Joel J. P. C. Rodrigues, Gustavo F. A. Gomes, Pedro Almeida, Ricardo A. L. Rabelo, Neeraj Kumar, Shahid Mumtaz, "An IoT-Based Solution for Smart Parking," Lecture Notes in Networks and Systems, Springer Singapore, 2020, pp. 213–224. DOI: 10.1007/978-981-15-3369-3_16.
4. Sinara P. Medeiros, Joel J. P. C. Rodrigues, **Mauro A. A. da Cruz**, Ricardo A. L. Rabelo, Kashif Saleem, Petar Solic, "Windows Monitoring and Control for Smart Homes based on Internet of Things," in *4th International Conference on Smart and Sustainable Technologies (SpliTech 2019)*, Split, Croatia, June 18-21, 2019, pp. 1-5, DOI: 10.23919/SpliTech.2019.8783163.

1.6 Thesis statement

IoT middleware is crucial in IoT solutions given the diversity and complexity involved in IoT environments. Various open-source middleware solutions are not designed with scalability in mind and do not address critical security considerations because they can imply less usability and reduced performance. Since middleware handles precious data and should be a secure location for human users, applications, and devices, scalability and security should be top priorities. Also, middleware are located in powerful servers, which means that they can ensure scalability and security without sacrificing too much performance, as long as the chosen programming language is appropriate, the architecture is scalable, and optimizations that take advantage of aspects such as self-contained security keys and other aspects presented in this thesis. Furthermore, any aspect that is related to security should never be neglected, especially in a software with a crucial role like the IoT middleware.

1.7 Document organization

The remainder of this thesis is organized as follows. Chapter 2 provides an overview regarding the IoT landscape, showing the different types of IoT device classification according to the processing power and communication capabilities, the

most popular ways of connecting IoT objects, the most common message exchange patterns on the Internet and why PubSub (Publish/Subscribe) is so popular in the IoT. Next, IoT middleware are introduced, their importance in IoT environments is highlighted, especially when it is so difficult to enforce global standards. Then, reference architecture IoT Middleware modules is presented, as well as the main issues with the existing solutions.

Chapter 3 overviews shows most popular qualitative and quantitative evaluation metrics, their usage in previous studies as well as the obtained results, and proposes the usage of Multi-criteria decision making techniques to combine these various metrics in the same comparison study and identify the best solution according to a chosen scenario.

Chapter 4 studies the importance of finding alternatives for devices that are incompatible with the chosen middleware solution because it does not support a certain application-layer protocol and proposes MiddleBridge, a simple, yet efficient solution that can be applied in such cases.

Chapter 5 discusses the influence of the underlying programming language in the performance of an IoT middleware.

Chapter 6 proposes, describes, demonstrates, and validates In.IoT, a middleware solution that was built based on the previously mentioned reference architecture, the solution provides innovations regarding security and usability, while also performing better than the other solutions under heavy load.

Chapter 7 proposes, describes, and demonstrates, OLP, a low-code platform that allows users with no coding background to build custom applications that can be tailored to specific IoT scenarios. The applications developed using OLP can consume the of middleware solutions such as In.IoT.

Chapter 8 proposes, the use of XGBoost, a machine learning technique to detect the replication attack, where attackers obtain device credentials and use to disturb the IoT network. A public dataset was used to train the model and the experiments show that the solution has great utility and accuracy.

Chapter 9 concludes the thesis elaborating on the main conclusions of this study, the learned lessons, and suggestions for further research works.

2 Related Work

The Internet of Things (IoT) is occasionally referred as the Internet of Everything (IoE) [41], while others consider the IoE term an expansion of the IoT concept that is much broader and could include even humans acting as devices [42][43]. For the purpose of this thesis, IoT and IoE will be treated as synonyms. Since its inception, the term has experienced minimal modifications and the fundamentals are still the same. IoT can be described as a scenario where everything may be inserted in a network, be uniquely identified, and interact with minimal human intervention [44][45].

The things may belong to the physical world in the sense that they occupy a mass, or the virtual world (virtual "things") that is only present in a simulation ecosystem [46]. To simplify, if users or applications have access to the information and communicate with these things (objects) through the Internet, it can be considered IoT scenario. The IoT concept is often associated with wearables [47] and smart homes [48], but it can be applied in any other environment, such as industry [49], healthcare [50], smart cities [51], smart cars [52], and many more. The flexibility of IoT across various environments is illustrated in Figure 1, which shows the IoT landscape in different verticals.

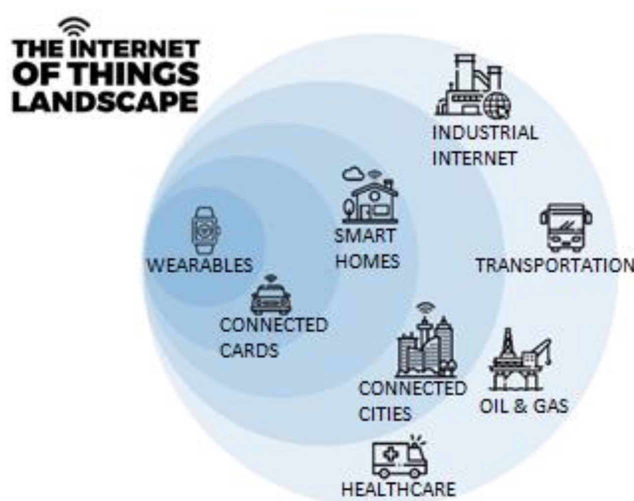

Figure 1 –*The IoT landscape across various verticals.*

History demonstrates that, as the physical size and cost of technologies diminish, more individuals gain access to them and, consequently, the presence of such technology becomes abundant in daily life [53]. Therefore, IoT devices should become increasingly popular throughout daily life as time goes by, especially when in 2016, 84% of the world population was already living in areas where Internet services are offered [54].

Since the IoT concept revolves around devices and the data they generate when interacting with human users, external applications or even other devices, understanding how the various devices can be classified, enables developers to build them without increasing costs. One of these classifications is displayed in recommendation ITU-T Y.4460 covers device classification regarding processing and communication capabilities [36].

2.1 Device classification regarding processing capabilities

The processing capability defines how the devices can perform computational tasks and execute algorithms. In this sense, the device processing capabilities can be classified as no processing, low processing, and high processing [36].

Devices with no processing capabilities have no processing capabilities to execute any actions, which means they are passive and extremely low-cost, with no embedded microcontrollers. RFID (Radio Frequency Identification) and NFC (Near Field Communication) tags are good examples of this type of device that could be applied to disposable packages [55]. These devices have no processing capabilities because the microcontroller will often be more expensive than the package itself.

Devices with low processing capabilities have sufficient processing capabilities to read or write data from sensors or actuators and send or receive the data as messages to IoT applications. These devices do not have enough processing capabilities to make decisions or run complex algorithms. They are generally low-cost devices with very limited microcontrollers, making the product economically viable. Smart lights and door sensors are an excellent example of this type of device because they count on other cloud services for data storage or data processing. The low processing capabilities are justified because a more powerful microcontroller could make the product more expensive, and the microcontroller would be underused.

Devices with high processing capabilities have enough processing capabilities to make decisions and run complex algorithms. They also possess the ability to coordinate other devices directly. Usually, these are high-cost devices, which means that they will have an embedded operating system. Devices capable of holding Virtual personal assistants such as Alexa from Amazon [56], Cortana from Microsoft [57], Siri from Apple [58], and Google assistant [59] are examples of this type of device.

2.2 Classification regarding communication capabilities

Communication capabilities specify how the devices connect to the communication networks, and IoT devices should possess communication capabilities. A device can be classified as low connectivity or high connectivity capability device according to its communication capabilities [36].

Devices with low connectivity capabilities do not implement an IP stack or any other NGN stack, which means they are not directly connected to the Internet. These devices communicate with cloud services is through an intermediary such as a gateway that possesses Internet connectivity.

Devices with high connectivity capabilities implement an IP stack or any other NGN stack, which means they are directly connected to the Internet. These devices communicate directly with cloud services withouth needing an intermediary such as a gateway because they already possess Internet connectivity.

2.3 IoT device classification regarding processing and connectivity capabilities

The processing and communication capabilities classification can be combined, resulting in three types of devices, namely: *i*) low processing and low connectivity (LPLC) device; *ii*) low processing and high connectivity (LPHC) device; *iii*) high processing and high connectivity (HPHC) device [36]. Other classifications can be derived, in the sense that, in theory, devices with no processing capabilities can have low or high connectivity, and high processing devices can have low connectivity. However, these other classifications were not considered because devices with no

processing capabilities such as RFID and NFC will mostly be used to identify objects in transit and will be discarded after its use. Regarding devices with high processing and low connectivity (HPLC), although possible, they are unlikely to exist in IoT environments and will not be considered.

2.3.1 Architectural reference model for LPLC devices

Low Power Low Connectivity (LPLC) devices simply act as an interface to collect data from physical things or the surrounding environment, and/or perform operations on physical things or the surrounding environment. These devices do not have sufficient processing capabilities to make decisions or run complex algorithms; they also do not have sufficient connectivity capabilities to directly connect to the communication networks (i.e., they do not implement an IP stack). For these reasons, a gateway is needed to act as an intermediary between these devices and the IoT (e.g., cloud services and applications). Figure 2 shows the architectural reference model for an LPLC device. In this reference model, the message handling and gateway access functional entities are the core functional entities.

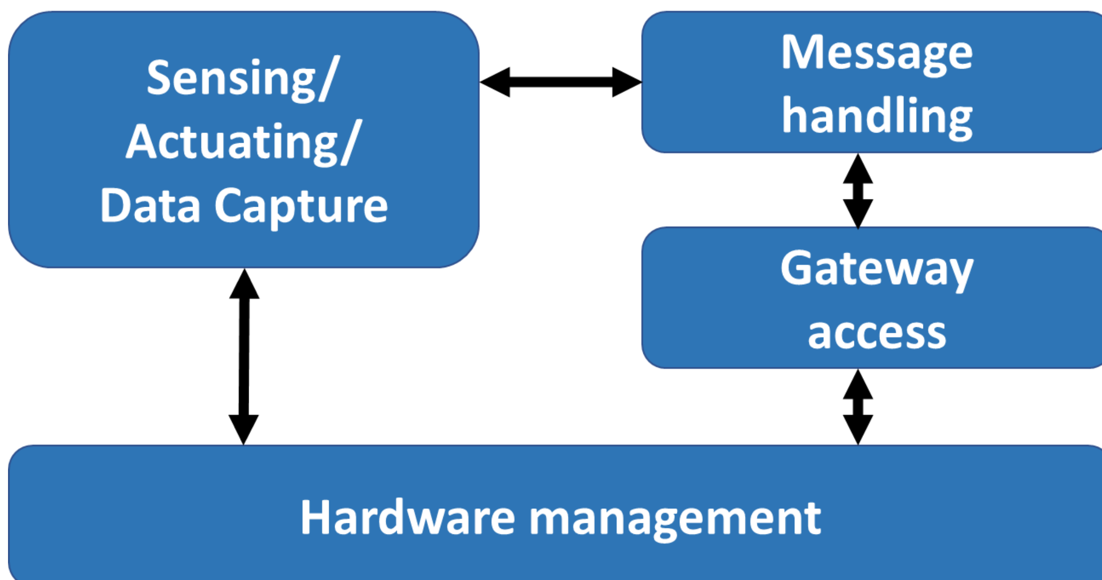

Figure 2 – *Architectural reference model for Low Power Low Connectivity (LPLC) devices.*

The sensing/actuating/data capture functional entity provides functions to read data from sensors, write data to actuators and capture data from data-carrying devices or data carriers attached to physical things. The message handling functional entity provides functions to send and receive messages, by using an application layer protocol. It also can provide a state machine for handling incoming messages. The gateway access

functional entity provides functions for communication management with the gateway. The hardware management functional entity provides functions for accessing the hardware (sensors and/or actuators, physical communication interfaces, hardware peripherals such as timers, analogue-to-digital converters (ADCs), etc.).

2.3.2 Architectural reference model for LPHC devices

Low Power High Connectivity (LPHC) devices have enough connectivity capabilities to directly communicate with the Internet (i.e., they implement an IP stack). Thus, there is no need for gateways mediating the communication between the devices and the applications or cloud services. However, devices still do not have sufficient processing capabilities to make decisions or run complex algorithms. Figure 3 shows the architectural reference model for an LPHC device. In this reference model, the gateway access functional entity is exchanged by a connectivity management functional entity. There is also a cloud service/application interface functional entity, that is responsible for understanding the application layer protocols used by the cloud service or application and its application programming interfaces (APIs) for sending/receiving messages and performing cloud services or applications operations.

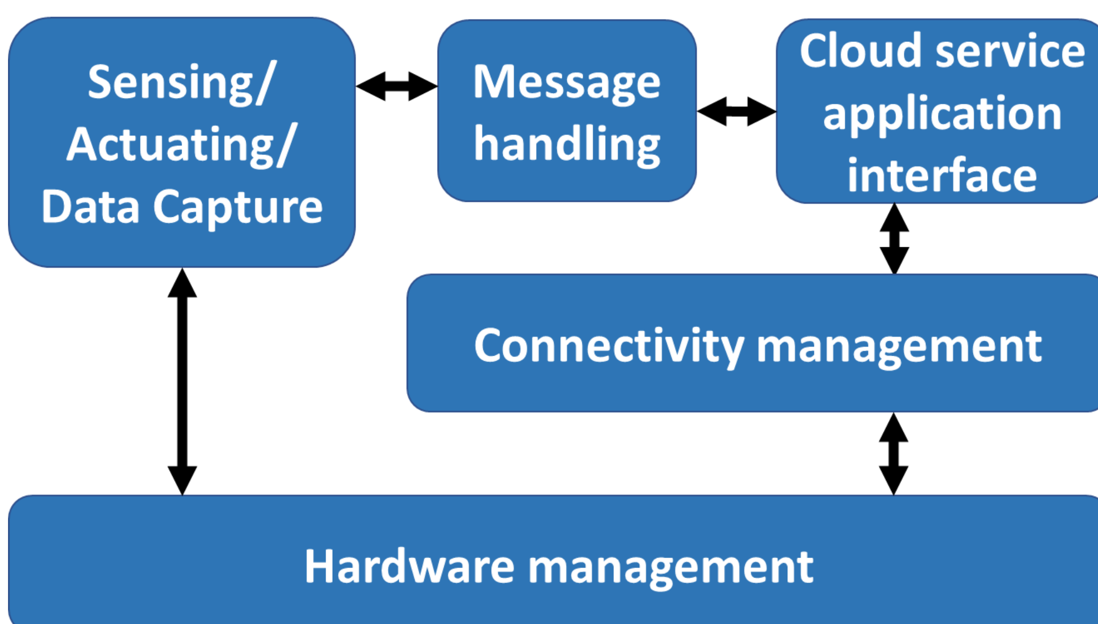

Figure 3 – *Architectural reference model for Low Power High Connectivity (LPHC) devices.*

The cloud service/application interface functional entity provides functions to interact with the IoT cloud service or IoT application, send and receive messages to the

IoT cloud service or IoT application, register/authenticate the device, etc. The connectivity management functional entity provides functions for connectivity management between the device and the communication network. The remainder entities were previously described in Section 2.3.1.

2.3.3 Architectural reference model for HPHC devices

High Power High Connectivity (HPHC) devices not only have high connectivity capabilities, making them able to directly connect to applications and cloud services, but also sufficiently high processing capabilities to make decisions and run complex algorithms (e.g., artificial intelligence (AI)-related algorithms). These devices are autonomous; they make decisions about their own functions and can also coordinate other devices. Figure 4 shows the architectural reference model for an HPHC device. In this reference model, the application execution engine functional entity is the core functional entity, providing application execution capabilities and interacts directly or indirectly with all other functional entities. Besides the core functional entity, other functional entities in the architectural reference model are often commonly used.

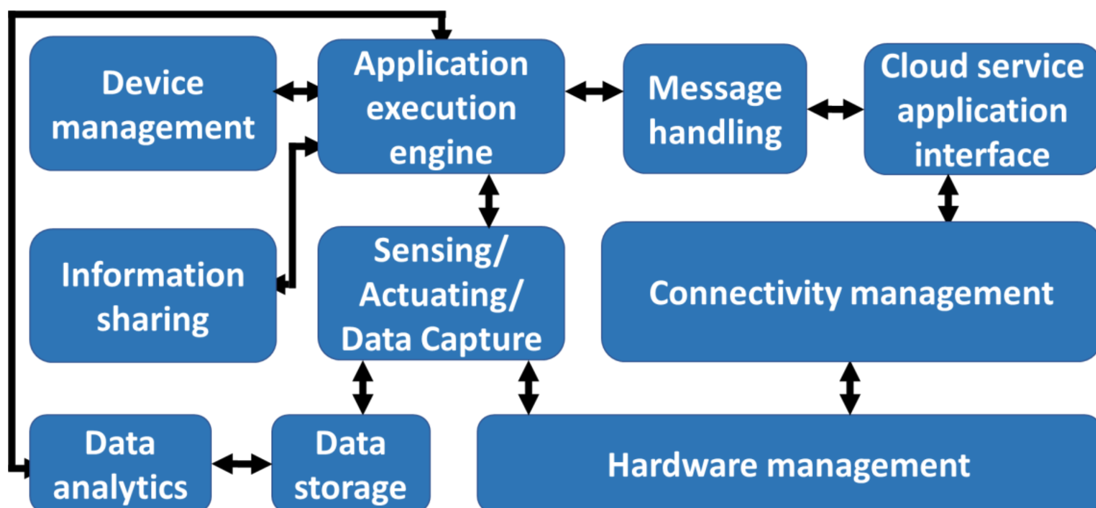

Figure 4 – *Architectural reference model for High Power High Connectivity (HPHC) devices.*

The application execution engine functional entity provides functions to install, delete, update and run applications on devices. Provides to applications access to other functional entities. The device management functional entity provides functions to manage other devices connected to the device and the device itself. The information sharing functional entity provides functions such as device to device interaction (data

exchange between devices), service discovery, service monitoring and service discovery interoperability. The data analytics functional entity provides functions for data processing and autonomous decision by running analytics and AI algorithms. The data storage functional entity provides functions of data storing and retrieving. The remainder entities were previously described in Section 2.3.1 and 2.3.2.

2.4 Connecting objects to the Internet using IoT

Most IoT devices will use wireless technology to access the Internet, and the most popular wireless technology is Wi-Fi (IEEE 802.11), which is available in most households and does not take IoT limitations into account. For this reason, alternative technologies are being utilized on IoT environments, such as the Bluetooth 5 and the IEEE 802.15.4 that is part of both ZigBee and 6LoWPAN (IPv6 over Low Power Wireless Personal Area Networks) protocol stack.

Bluetooth 5 is the latest iteration of the popular Bluetooth standard and similar to Bluetooth 4.2, Bluetooth 5 also supports IP networks [60]. Unfortunately, users rarely explore the IP capabilities provided by Bluetooth, and it is currently on version 5.2. Bluetooth is also popular in IoT implementations in the form of BLE (Bluetooth Low Energy), which was specified in version 4.0 [61]. IEEE 802.15.4 is a standard for Low-Rate Wireless Personal Area Networks (LR-WPANs) that specifies the physical and MAC layers of the OSI model [60]. 6LoWPAN and ZigBee deployments use IEEE 802.15.4. 6LoWPAN is an Internet Engineering Task Force (IETF) approach that compresses and encapsulates the IPv6 headers and accommodates them on the frame IEEE 802.15.4 [60]. ZigBee was developed and maintained by the ZigBee Alliance, and it is mostly known for its mesh topology, but it supports other topologies, such as star and tree [60]. The issue with IEEE 802.15.4 technology is that its performance is impacted by W-iFi interference [62], and Wi-Fi is predominant in most urban scenarios.

Connection to the Internet through 3G/4G (and very soon, with 5G) networks is the other most popular method because of its high availability in urban scenarios but has the similar issues to Wi-Fi when it comes to the Internet of Things. For this reason, long-range network technologies such as LoRa [63], Sigfox [64], and IEEE 802.11 ah (HaLow) [65] were developed [66]. Their description is in the name, less battery

consumption and broader area coverage. LoRa and Sigfox need a gateway that interacts with the end devices, the gateway is connected to a backhaul that provides Internet connectivity. Sigfox operation is similar to traditional ISPs (Internet Service Providers), where users pay for the service they are subscribed, while LoRa offers infrastructure that can be purchased and installed by any user, which can then use the network at will. The advantage of IEEE 802.11ah over the other LoRa and Sigfox is that its end devices natively supports IP networks [67][68].

Another promising method of accessing the Internet through IoT is 5G technology, that reached end-users in 2020 and like its predecessors is expected to become mainstream. 5G presents difference performance requirements for distinct scenarios and IoT is one of them [69]. Despite 5G deployments not being mainstream, 6G is already in development, and could bring innovations with a super Internet of Things [70].

2.5 Message exchange patterns

When computers communicate, they establish a connection through an application layer protocol and exchange data through a message exchange pattern. The most popular message exchange patterns are request-response (RR) and publish/subscribe (PubSub). RR is common in distributed systems, and it consists of a sender requesting a resource. The receiver (generally a server) sends a reply corresponding to the resource in a unicast communication [71]. If the server cannot attend to the request, the server replies with an error message. This concept is predominant when browsing the Internet and is one-to-one communication.

The PubSub paradigm consists of subscribing to events, in the sense that when a subscriber enrolls in a certain topic, he automatically receives a copy of the published message [71]. In PubSub, the server that stores the user subscriptions and accepts the publications is called a broker. When a new event arrives, the broker forwards a copy to the subscribers, which automatically receives it. In most of the PubSub implementations, only the broker has access to subscribers' identity and number. The PubSub paradigm is essential for any IoT middleware because it allows simple interaction between devices. The most known implementations of the PubSub paradigm

are Apache Kafka, which is topic-based and can scale [72], RabbitMQ that supports PubSub and request-reply [73], and Mosquitto, a broker for the MQTT protocol [74].

As previously mentioned, besides the message exchange pattern, computers must establish a connection through an application layer protocol. One of the most popular application layer protocol is HTTP (Hypertext Transfer Protocol) [75]; it supports both PubSub and RR paradigms. An example of PubSub with the HTTP protocol is the Orion Context Broker (a Fiware project) [76][77]. HTTP usage is discouraged in IoT scenarios because it consumes too many resources from the constrained devices [78]. Consequentially, alternative application layer protocols such as CoAP (Constrained Application Protocol) and MQTT (Message Queuing Telemetry Transport) are increasingly popular in IoT scenarios [79].

MQTT is a PubSub protocol and uses TCP (Transmission Control Protocol) at the transport layer [80]. CoAP is based on a REST (Representational State Transfer) architectural style and uses UDP (User Datagram Protocol) at the transport layer. CoAP is a lighter HTTP in the sense that it also possesses the POST, PUT, GET, and DELETE methods [81], and most implementations use the RR paradigm. Although CoAP is more efficient than MQTT regarding latency and packet size, the MQTT protocol is much more popular. This popularity is likely because MQTT allows multiple users to be notified when a device sends a message (because of the PubSub paradigm). CoAP recognizes this increase in MQTTs popularity, and currently, some implementations use PubSub, and a CoAP draft that supports PubSub is currently under revision by IETF (Internet Engineering Task Force) [82].

The solutions presented in the thesis use both message exchange patterns because they are relevant in most IoT scenarios.

2.6 IoT Application layer protocols

An application layer protocol determines how clients operate in different systems and exchange messages among them. Several application layer protocols were proposed for IoT, being the most notable MQTT and CoAP. The MQTT protocol was standardized in 2013 by the Organization for the Advancement of Structured Information Standards (OASIS) [83]. MQTT is based on a publish/subscribe

communication model and uses TCP (Transmission Control Protocol) at the transport layer [80]. CoAP is specified in RFC 7252 [84] and is based on a REST architectural style, with a client-server architecture. It is generally described as a lightweight HTTP and uses UDP (User Datagram Protocol) at the transport layer [80], which means it is a stateless protocol [85]. Other relevant solutions include DDS, XMPP, and Websockets. DDS was standardized by the object management group (OMG) [86]. It is based on a publish/subscribe model [87], follows a peer-to-peer architecture, and is mostly used for reliable and efficient real-time communications [88].

In contrast to other publish/subscribe protocols, the DDS architecture is "brokerless" [89][90]. The downside of DDS comes from the fact that its specification does not determine which protocol should be used at the transport layer, stating that different implementations must provide "vendor-specific bridges" to interoperate [91]. XMPP was initially called Jabber [92], and it is specified at the RFCs 6120 [93] and 6121 [94]. It uses a client/server architecture and was developed as an instant messaging protocol that allows users to chat on the Web in real-time [95]. Most XMPP implementations are attached to a graphical chat client (useless for most IoT devices). Furthermore, the protocol development seems stagnated, and Google ceased support for a standard [96] and its usage in IoT is reduced. Websockets were standardized in 2011 by the IETF (Internet Engineering Task Force) through the RFC 6455 [97] and uses TCP at the transport layer [98]. Websockets are useful for IoT because they provide full-duplex communication through a single TCP connection [99].

Regarding possible deployment of the above-described protocols, XMPP and DDS are lackluster. As previously mentioned, XMPP active development seems to be stagnated, and DDSs has few implementations. In contrast, MQTT's popularity is increasing, mainly because the community is very active in online forums. Also, various deployments of the protocol are available for numerous platforms, and the protocol is easy to use and understand. CoAP has many implementations, but the community is not so active as MQTT. It is important to note that despite its inefficiency in IoT scenarios, REST remains very popular, mainly because it is well documented, and various examples are available online. Concerning efficiency, protocols that use UDP are more sensitive to packet loss, and TCP produces more overhead as a tradeoff for the increased reliability.

In [100], the performance of MQTT, CoAP, and DDS (TCP version) was compared and revealed that DDS consumes at least two times more bandwidth than

MQTT, followed by CoAP. Also, under degraded network conditions, MQTT experiences higher latency, followed by DDS and CoAP. In [101], the performance of MQTT, CoAP, and Websockets was compared under normal network conditions (no packet loss), revealing that MQTT produces more overhead, followed by Websockets and CoAP. Concerning latency, CoAP performed better, followed by Websockets and MQTT. Overall, the conclusions of both studies are consistent and allow this thesis to infer that DDS produces larger packets, followed by MQTT, Websockets, and CoAP. Regarding latency, CoAP performs better, and MQTT presents the worst performance. There is not enough data to determine if WebSockets performs better than DDS regarding latency. Table 1 summarizes the comparison for the Websockets, DDS, CoAP, and MQTT protocols.

Table 1 – *Summarized comparison among Websockets, DDS, CoAP, and MQTT protocols.*

Comparison criteria	Comparison results (Best to worse)
Less bandwidth consumption [100]	CoAP; MQTT; DDS
Less latency under degraded network conditions [100]	CoAP; DDS; MQTT
Less overhead with no packet loss [101]	CoAP; Websockets; MQTT
Latency with no packet loss [101]	CoAP; Websockets; MQTT
Less Packet size (inferred from [100] and [101])	CoAP; Websockets; MQTT; DDS
Less latency (inferred from [100] and [101]	CoAP; MQTT; DDS / Websockets ***

*** – There is not enough data to determine whether WebSockets performs better than DDS regarding latency.

In Chapter 4, the MQTT, HTTP, DDS, and websocket protocols were used by the MiddleBridge solution. In Chapter 6, a solution called In.IoT is presented, it supports the MQTT, HTTP, and CoAP protocols.

2.7 A global IoT standard

IoT environments will have various devices from different brands and vendors. Unfortunately, most of these devices are incompatible with gadgets from partner brands. This compatibility issue could be solved through a common standard, but enforcing such a standard is difficult because of price, politics, implementation complexity, geography, usability, or even lack of complementary technologies. From a purely technical standpoint, the biggest issue reaching a common standard is that every standard has its benefits and drawbacks. From a geography standpoint, a particular standard is sometimes better suited for a region because of its geography, climate, or spectrum

distribution. It is not easy from a price standpoint because the best solution is sometimes costly and cannot be widely adopted. The pricetag is one of the reasons ATM lost to ethernet since ATM was in some aspects superior to the ethernet technology [102].

From the usability standpoint, it is difficult because the standard has to reach the end-users in a product. If the product is deemed too difficult to use, the end-user will not recommend adopting the product. Another issue with a common standard is that sometimes a standard does not gain traction because few technologies complement it. Take the MQTT protocol, which was invented in 1999 [103] but only started to gain traction with the increased interest in the IoT concept. From a political standpoint, it isn't easy because it is common for countries and big corporations to invest in research, and sometimes the investment returns by selling the technology when it is ready to use. Politics also plays a significant role in the sense that there are several standardization initiatives. In IoT for example, there is OneM2M [104], OpenFog [105], and many more trying to develop an efficient and sustainable IoT.

Besides the mentioned initiatives, other established standardization entities, such as 3GPP (3rd Generation Partnership Project) and IEEE (Institute of Electrical and Electronics Engineers), are also developing IoT standards. With so many standardization entities, the standards' differences are well-document, and standards are often backed by the mentioned entities. However, it is common for standardization initiatives to support competitor standards. The Open connectivity foundation endorses Alljoyn [106] and IoTivity [107], even though both address device connectivity (Alljoyn later merged with IoTivity [108]). Also, it is common for the less known standardization initiatives to merge with other standardization bodies. Such examples include IPSO alliance merging with the OMA alliance to form OMA Specworks [109] and Allseen Alliance merging with Open Connectivity, maintaining the Open Connectivity name [110].

With so many bodies developing competing and often similar standards, other issues can emerge, even when a common standard is established and enforced. What happens if said standard is not scalable? What if a superior standard is later developed? These questions are being formulated in one of the best examples of standardization in human history, the current Internet. Some argue that major modifications or even a complete revamping of the current Internet architecture is needed because it reached a scale that was never imagined during its inception, at the cost of several mendings that are no longer sustainable [111][112]. Therefore, interoperability among devices through

a universal standard is difficult, and even if it is achieved, the standard might only attend to the needs of the IoT that is envisioned in current days, which might not scale well in the future. With the absense of a global standard, the IoT middleware assumes an increasingly important role in IoT environments.

2.8 IoT Middleware

An IoT middleware is a software present in nearly every IoT scenario because it collects data that is sent by devices and allows them to communicate and act upon such data [16][113]. This software can be applied in every IoT scenario because they allow more complex IoT applications to be built on top of them. Most middleware are generic and provide basic functionalities such as data insertion and consultation [114]. However, some are built for specific areas such as industry and smart homes, providing a rich graphical user interface and various scenario-specific functionalities. A specific approach's disadvantage is that it can only be applied in scenarios that are similar to its original inception. IoT middleware is also known as IoT platform, IoT middleware platform, or simply middleware [15].

A real-life analogy to IoT middleware's role is a translator in a scenario where three individuals from different nationalities debate. If they do not use a common language (the standardization option), they need a translator mediating the conversation. Now imagine that the three individuals are distinct applications (APPs). APPs generally communicate through REST APIs (the language) over the Internet, and generally, the Apps will have different REST APIs. Without a middleware (the translator) each APP must understand every other REST API, which is increasingly complicated as multiple APPs and devices are expected in IoT environments. This simple idea allows users to focus on the problem and is illustrated in Figure 5, because instead of knowing how each application works, users manipulate data from one application (the middleware).

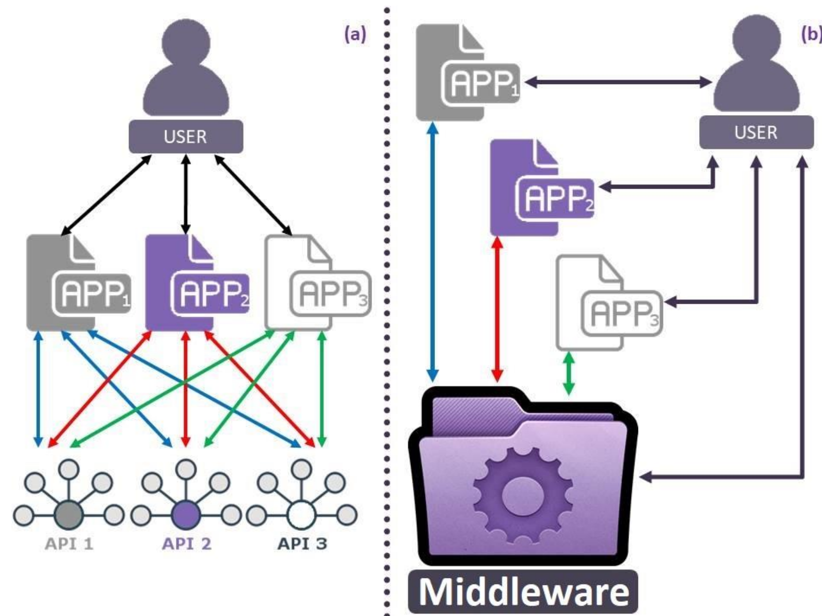

Figure 5 – *Illustration of the communication (a) without middleware and (b) with middleware.*

Currently, the most relevant open-source middleware solutions are Sitewhere, Linksmart HDS, Konker, and Orion (a Fiware project). They are considered the most relevant, because they were evaluated in one of the most broad performance evaluation studies in the literature that was performed by Cruz et. al. [16] Other open Middleware solutions are Dojot [115], ThingsBoard [116], Devicehive [117], Ubidots [118], Xively [119], and many more are created each day.

Orion (a Fiware project): It is common, even among the scientific community to call Fiware an IoT middleware, when actually, Orion Context Broker is the IoT middleware solution. Orion is an open-source IoT platform maintained and created by the European project Fiware and is licensed under Affero General Public Licence (GPL) version 3 [120]. It is a publish/subscribe implementation of the NGSI-9 and NGSI-10 Open RESTful API specifications, and only supports HTTP RESTful communications. To successfully deploy Orion on a server, users must install MongoDB, because it is used as the database engine where data is stored [121]. The downside of Orion is that as a message broker, the specification states that only the last received value is kept in the database, which means that Orion does not support historical data consultation. Recognizing the limitations of Orion, STH (Short Time Historic) and Cygnus were developed by Fiware, as well as other so called IoT agents that support other protocols such as CoAP and MQTT [122]. Cygnus and STH function very similar with Orion, in

the sense that both subscribe to Orion notifications, and as soon as values are received, they are persisted to the database. Fiware officially supports both Cygnus and STH and the main difference between them is that Cygnus only stores data, and no consultation is possible, while STH allows both. Orion is currently in “version 2” of its REST API. The underlying programming language that was used to build Orion was C++. More information regarding Orion and STH can be found in their official documentation [123][124].

Sitewhere is an open-source IoT platform maintained and created by Sitewhere and is licensed under CPAL-1.0 (Common Public Attribution License Version 1.0). Despite being open-source, they offer a paid version that offers other benefits such as dedicated support. It supports MQTT, AMQP, and HTTP REST as application-layer protocols [125]. Sitewhere moved to the Microservices architecture [126], and version 3 is already in development. The underlying programming language that was used to build Sitewhere was Java. More information regarding Sitewhere can be found in their official website [127].

Linksmart HDS, formerly known as Hydra [128], is an open-source middleware platform that is licensed under Apache license 2.0. It supports REST communications with its server, and data visualization is made through Grafana. To successfully deploy the solution, users must have either influxDB or MongoDB installed [129]. Another option is to run Linksmart using Docker. It is one of the few platforms that uses SenML instead of JSON to represent data. The underlying programming language that was used to build Orion was the Go programming language from Google [130]. Go is a relatively new programming language, its version 1.0 was released in March 2012 [131]. More information regarding Linksmart can be found in their official documentation [132].

Konker is an open-source middleware platform created and maintained by the Brazilian KonkerLabs. It is licensed under Apache license 2.0. Although it is an open-source, an online version is available as PaaS where users can trial for free, or expand to a paid version that offers other benefits such as dedicated support. It supports REST and MQTT as application-layer protocols. To successfully deploy the solution, users must have MongoDB and Cassandra. Another option is to run Konker using Docker. It

uses Java as the underlying programming language. More information regarding Konker can be found in their official website [133].

The mentioned middleware solutions will be used throughout the thesis in performance evaluation studies.

2.9 IoT middleware reference architecture

Most IoT scenarios are only possible through the usage of IoT middleware, which gathers data from gadgets and acts upon such data. Given the resource constraints on most devices, which can only execute simple actions, the role of the middleware is crucial. Recognizing the importance of middleware in IoT scenarios, the work of Cruz *et al*. [15] proposes a reference architecture for IoT middleware solutions. This architecture envisions a platform composed of six modules, which consider requirements such as scalability, reliability, event management, security, resource discovery, and many more. The proposed modules are *i*) Interoperability, *ii*) persistence and analytics, *iii*) context, *iv*) resource and event, *v*) security, and *vi*) Graphical User Interface (GUI). The reference architecture is illustrated in Figure 6.

The **interoperability module** exposes functionalities through an API, which should support as many application layer protocols and data representation methods as possible. This module also suggests that it should provide SDKs (Software Development Kit) in various programming languages, as developers are more likely to use tools in their favorite programming language. Furthermore, it should support the most common data representation methods, such as XML (eXtensible Markup Language) and JSON (JavaScript Object Notation).

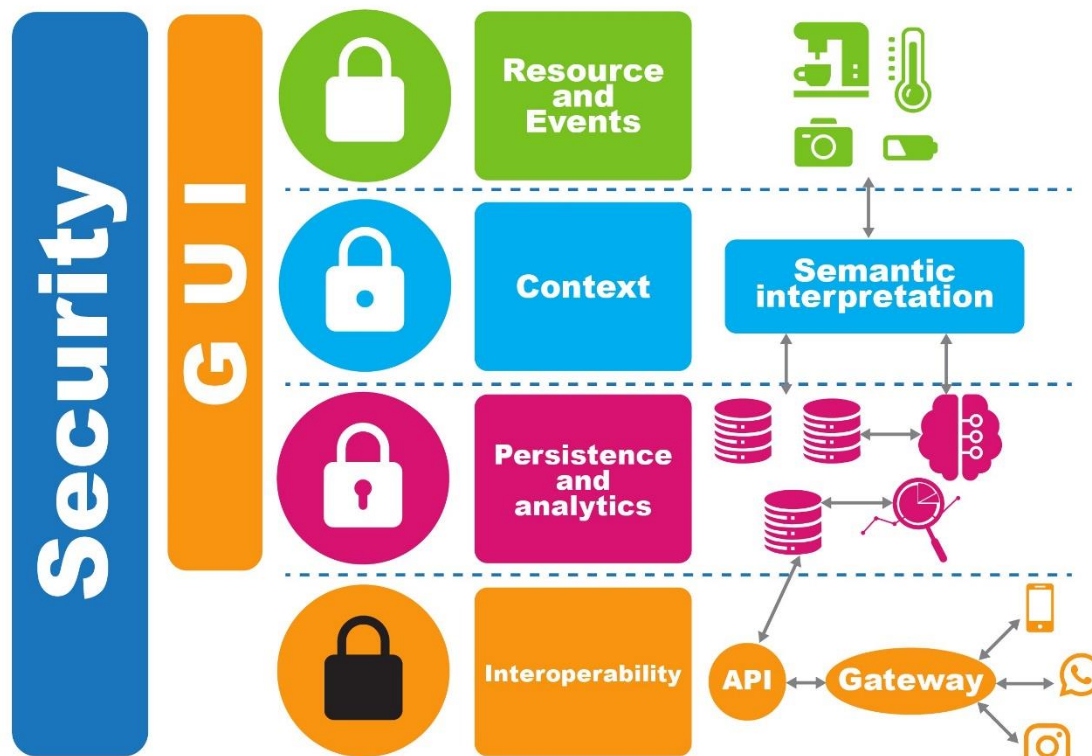

Figure 6 – *IoT middleware reference model.*

The **persistence and analytics module** consists of data storage and management. The module suggests that NoSQL databases should be used instead of SQL databases because more data will be stored in IoT environments than consulted, and data insertion is faster in NoSQL databases [134][135]. The module also recommends to process data through data analysis techniques or feed it to machine learning algorithms. These techniques could reveal hidden user patterns and contain valuable insights. Additionally, basic data consultation mechanisms, as well as basic analytics, should be provided.

The **context module** suggests that the platform should be aware of the context in which the device is sending the data to achieve the envisioned smart IoT. For this reason, artificial intelligence (AI) might even be necessary, or at least the use of external APIs that consult an AI.

The **resource and event** module states that devices should know which actions they can perform so that they can be advertised to others as well as be discovered. It recommends devices and applications to announce their capabilities. Then, authorized

devices can query for the service with relative ease. Another important feature should be the facilitation of firmware updates by the middleware. Updates are usually performed manually by the user for each device. However, in a scenario with thousands (maybe millions) of devices, this is not a scalable approach.

The **graphical user interface** (GUI) module recommends every middleware solution to provide an administrative dashboard to manage and visualize their device data through a graphical interface. Since various applications will be built on top of middleware, good APIs are more important than GUIs. However, a basic GUI is a must in the modern world.

The **security** module states that solutions should be easy to use without compromising security, mainly because this software holds precious data, which should only be disclosed to authorized users. In this sense, the reference architecture suggests four (4) security features, namely: "*i*) Individual device credentials, *ii*) Devices should use different credentials to publish and consult data, *iii*) devices should access other device data using their credentials, and *iv*) middleware should know device habits and store their MAC and IP".

Every device should have its unique credentials to assure the middleware data's safety, hence the recommendation for **individual device credentials**. This feature is essential because it guarantees that if credentials are compromised, the impact is limited to the affected device, and the user can change the device credentials. The second recommendation states that **devices should use different credentials to publish and consult data**. This will allow organizations to expose the device data to external users without compromising security. The third recommendation states that devices should be able to consult other device data if they are authorized to perform such action. This is the only way they can efficiently share data without exposing their credentials. The fourth recommendation states that **middleware platforms should be aware of device habits and store additional information, such as the device IP address when it sends data**. This final recommendation would help identify security breaches, as anomalies such as sending data in irregular intervals, accessing different content, or an IP address from a different region, are indicators that something might be wrong. If the middleware can detect these anomalies, it should alert the users. This final security feature can only be overturned by an advanced attacker, aware of the security features that can even disable the primary device (or the middleware would detect duplicate publication).

The most relevant open-source middleware solutions are Sitewhere, Linksmart, Konker, and Orion (a Fiware project). They are considered the most relevant, because they have been the subject of several performance evaluation studies. The biggest issue is that they do not provide individual credentials for the devices, and the one with such a feature (Konker), does not use any authorization mechanism. Sometimes, the lack of individual credentials is also valid for PaaS solutions (Platform as a Service) that are only available in the cloud. Another issue with most solutions is related to the MQTT protocol, which does not enforce restrictions on which topics can be accessed after a successful authentication to the broker.

Moreover, most solutions do not offer an easy way to integrate new devices into the platform, this is likely the reason for the shared credentials. The issue with this approach is that all the devices have the credentials of the admin user and elevation of privilege is a real threat. Besides, despite being open-source, various solutions are poorly documented or offer no tutorials for developers, and thus difficult to customize. Another issue is related to the response times when under heavy load. Regarding PaaS solutions, the biggest concern is that they can be terminated for various reasons, and even big names like Samsung Artik are not immune to this.

The reference architecture proposed by Cruz *et al.* [15] will set the ground rules for various aspects of the thesis, the most notable being In.IoT, a middleware solution that addresses various of the identified issues and is currently used in various IoT projects at Inatel. More details regarding In.IoT can be found in Chapter 6.

2.10 Authorization mechanisms for IoT middleware

Most web applications require users to provide their credentials, generally through a combination of username and password. This action is called authentication and allows the server to certify that the user is registered. After the authentication process, the server verifies which actions can be performed by the user, and this is called authorization. Authentication can also be simplified as an answer to “who are you?” and authorization as “what can you do?”. For security purposes, users should rely on authorization mechanisms that do not transmit their username and password in every request, to prevent malicious users from obtaining the credentials. All the authorization

techniques send a code to the user that symbolizes its successful authentication and is verifiable by the server. The most common authorization mechanisms are *i*) **Web session**, *ii*) **client tokens**, and *iii*) **third-party application access**.

A **Web session** is an authorization mechanism used by applications, in which the user stores a session ID (generally in a cookie) given by the server. The server stores the session ID and other user data that is relevant for the application in the RAM (Random Access Memory) to avoid additional database queries. Users can only access their corresponding session, and for security reasons, the session ID should be the only data that is stored on the client-side, since the client can alter it. The issue with Web sessions is that it can consume too much RAM when there are many users.

A **client token** is an authorization mechanism where the server generates a digitally signed token containing any user data, such as a username. This token is secure because the user can only deceive the server if he possesses the password that was used by the server to digitally sign the token. This technique also allows the server to not store any information regarding the session. The mainstream implementation of client tokens is JWT (JSON Web Token); the security of the JWT lies on the insurance that the user did not alter its data [136]. Therefore, hiding the data contained in the token is not one of the goals of a JWT, and such data is decoded without the secret. For this reason, only identifiers are encoded in the JWT, and sensitive information such as passwords are not placed inside the JWT. An issue with this technique is that the tokens cannot be revoked without changing the secret, which would affect all the users. Another approach could consist of restricting users from generating new tokens or even implementing additional validation on the server-side before attending the request. One disadvantage of JWT is that they require over 100 characters, which significantly increases the packet size.

Third-party application access is an authorization mechanism where registration in the application is not needed to access the desired features. Nowadays, some online services require the user to identify himself within the service, and filling the registration forms can take time. This issue is generally solved through third-party application access methods, the most famous being OAuth. OAuth is a standard that allows users to identify themselves within services through a third-party account (generally a social media account), without exposing the password to the service [137]. With OAuth, the user receives a token, generated by the social media account, and whenever the user communicates with the service, the token is sent. Before generating

the token, the user configures which of the user data will be accessible to the service. Whenever the service wants to access the user information, it will use the token provided by the user and only access what was previously authorized by the user. The issue with OAuth in IoT scenarios is that it would be difficult to notify the devices of any alteration in the token without sharing the user password with every device (assuming that each device would have a different token). Note that not only social media accounts are used for OAuth, they are just the most common occurrences.

A discussion on how the chosen authorization mechanism can impact the performance of the IoT middleware solution will be presented in Chapter 6.

2.11 Main issues of existing solutions

An IoT middleware is a software that is present in nearly every IoT scenario because it collects data that is sent by devices and allows them to communicate and act upon such data [16][138] . This software can be applied in every IoT scenario because they allow more complex IoT applications to be built on top of them. Most middleware are generic and provide basic functionalities such as data insertion and consultation, which means they must have at least, one database to store the devices' messages. Microservices architecture is not mandatory for an IoT middleware. Some middleware are built for specific areas, such as industry and smart homes, providing a rich graphical user interface and various scenario-specific functionalities. The disadvantage of a specific approach is that it can only be applied in scenarios that are similar to its original inception.

The biggest issues with the mentioned most relevant middleware solutions regarding performance under heavy load and security are: *i*) lack of individual credentials or authorization mechanisms for the devices, *ii*) lack of packet size optimization, *iii*) device variables are specified when a device is created, *iv*) lack of restrictions for the MQTT protocol, *v*) difficulty to integrate new devices into the platform.

- **Lack of individual credentials or authorization mechanisms for devices**: Despite it is an intuitive security feature, individual credentials for devices are not implemented to increase usability in many solutions. Regarding the previously

mentioned solutions, only Konker uses individual credentials, and even then, it does not use any authorization mechanism. From a security point of view, authorization mechanisms are important because once users authenticate, the server retrieves a code to the user, which allows them to send data to servers without transmitting their username and password in every request. From a performance point of view, if no authorization mechanism is used, the middleware must verify if the credentials are correct, then the message is stored. This is the reason why Konker's response times were lackluster in [16]. This can be solved by using an authorization mechanism that does not need a database consultation.

- **Lack of packet size optimization**: Smaller packet sizes generally result in less transmission time. The Ack message in Orion v1.8.0, which used NGSI v1, and Sitewhere v 1.11.0 that were studied in [16], has more Bytes than a message sent by the device. This is because they included the message sent by a device in the reply body. This issue was corrected in their recent versions. This was done to assure the device that the message was well-received and processed, but this is already the purpose of the reply code (ensuring a message was delivered and processed or not). If a device receives a successful reply code but notices an error because the reply message differs from the sent message, it is unlikely that a device sends a new message to correct this outcome. Another issue that affects packet size is the usage of reserved words to trigger events on the Middleware or even on other devices. Reserved words increase the packet size and force the Middleware to check their presence, slowing the data processing. Both these issues can be solved by following a minimalistic approach, where only data that is intended to be stored is transmitted on the message body. Regarding the reserved words, developers can increase performance by setting up specific routes used by the devices to trigger events, instead of processing all the messages the same way, expecting that some of them will trigger events.

- **Device variables are specified when a device is created**: When devices can only send variables that were specified upon its creation, whenever data is sent, the Middleware first verifies whether the variable was registered for that device. This issue affects Orion, but Linksmart also suffers from this issue. Linksmart performs so well for 1 (one) parameter and is lackluster afterward because a variable name is linked to device identification. This means that two separate devices cannot both send a variable named “temperature” since the device name is linked to its identification. When sending more than 1 (one) parameter, it must query each variable (each device) and store data

for each device. The first issue can be solved by not enforcing such strict rules when data is sent, and the second by unlinking the variable name with its identification.

- **Lack of restrictions for the MQTT protocol**: This is a prominent issue because, without it, any topic can be accessed after a successful authentication to the broker. A possible solution for this issue is proposed in Section 6.4.3.

- **Difficulty to integrate new devices into the platform**: Most solutions do not offer an easy way to integrate new devices into the platform. This is likely the reason for Sitewhere using shared credentials. With this approach, all the devices have the admin user's credentials, and elevation of privilege is a real threat. A solution to this issue is proposed in Section 6.5.1.

Other common issues are related to poorly documented solutions, which offer no tutorials for developers and thus difficult to customize. Regarding PaaS solutions, the biggest concern is that they can be terminated for various reasons, compromising the IoT environment's integrity, and even big names like Samsung Artik are not immune to this. For this reason, In.IoT was created, a solution that addresses several identified issues and is currently used in various IoT projects at Inatel.

2.12 Summary

The chapter provides an overview regarding the IoT landscape, showing the different types of IoT device classification according to the processing power and communication capabilities, the most popular ways of connecting IoT objects, the most common message exchange patterns on the Internet and why PubSub is so popular in the IoT. Then, some application layer protocols that are used in IoT are introduced (Websockets, DDS, CoAP, MQTT, and HTTP) and a table comparing the efficiency of these protocols (except for HTTP) is provided. Next, IoT middleware are introduced, their importance in IoT environments is highlighted, especially when it is so difficult to enforce global standards. Then, a reference architecture for IoT middleware is presented and its modules are detailed. Next, the most popular authorization mechanisms and their relevance in IoT environments is highlighted. Finally, the main issues with the existing middleware solutions is presented, as well as how they can be overcome.

3 Performance Comparison of IoT Middleware Solutions

The performance of an IoT solution is impacted by the chosen middleware, especially as the number of devices increases. Currently, there are a plethora of IoT middleware solutions with similar features. Choosing which middleware solution to deploy in a real IoT scenario is difficult because most of the available literature focuses on a qualitative comparison that can be subjective, instead of a quantitative comparison, which is often less subjective. In this sense, the data that is present in this chapter originates from the study by Cruz *et al.* [16], that uses qualitative and quantitative metrics to compare IoT middleware solutions.

3.1 Qualitative metrics for IoT middleware

Qualitative metrics mostly focus on the absence or presence of features. The most common qualitative metrics regarding IoT middleware are *i*) popularity, *ii*) supported application protocols, *iii*) quality of documentation, and i*v*) the presence of features.

Popularity: Tries to measure the level of adoption or general knowledge of the solution's existence. In scientific research, popularity can be measured by the number of mentions of the middleware project in scientific papers. In the industry's realm, this could be measured by the number of companies that adopt or back the solution as a sponsor. Another way of viewing this metric is the availability of a community that has experienced and overcome most of the bugs that are present in the solution.

Supported application protocols: The middleware will communicate with devices through application layer protocols. This means that a solution that supports more protocols will always have the advantage [139].

Quality of documentation: This metric considers the clarity of the available tutorials and primarily focuses on how easy it is to install and run the solution, but can be extended to documentation for more advanced used cases.

Presence of features: This metric considers the presence or absence of various features such as a graphical user interface for admin users or usage of a specific database. Additionally, features could extend to security. A popular qualitative metric involves measuring the presence or absence of security features such as individual device credentials or the distinction between admin users and device users.

The issue with a qualitative comparison such as those presented in [140], [141], [142], and [143] is that they are difficult to judge in a real-life IoT scenario because there is no way of knowing how the solution will perform when a high number of devices are present.

3.2 Quantitative metrics for IoT middleware

Quantitative comparison generally provides a more clear view, in the sense that numbers back it. The biggest issue with a fair quantitative comparison among middleware solutions is that it requires the same resources to be allocated to all the solutions being compared. This is especially troubling when many of the known platform solutions are only available in the form of PaaS (Platform as a Service) in the cloud, and it is impossible to determine what resources are allocated to it. For this reason, PaaS solutions are generally not included in quantitative comparisons.

The work presented in [138] performs a comparison between two publish/subscribe platforms (FIWARE and OneM2M/ETSI M2M). The quantitative metrics that the study considers are: **publish times for different number of parallel requests** (50 to 500 requests), the **goodput for different number of parallel requests** (unlike throughput, which measures all data that is transmitted, goodput measures the useful data), and the **number of retries throughout a day** (from 9:30 to 23:50). The study concluded that a broker performance depends on components such as the database and the underlying communication protocols.

In [144], a performance comparison between ThingsBoard and Sitewhere is made. The metric considered were **Througput**, **response time**, **CPU utilization**, **and active memory**, using 100 and 200 requests per second, reaching a maximum of 1000

concurrent users in all experiments. The study concluded that ThingsBoard presents better performance with HTTP, and Sitewhere was better for the MQTT protocol.

A broader study is presented in [16], which guides users attempting to compare IoT middleware solutions. The study presented in [16] recommends a qualitative comparison to filter which solutions are compliant with the proposed scenario and quantitative comparison to compare such solutions objectively. In the qualitative comparison, 11 open-source middleware solutions were considered. Then, the 5 solutions compliant with the proposed scenario progressed to a quantitative comparison where a simple scoring system was used. The solutions that perform better in a given criterion received 5 points, the second best received 4, and so on. The evaluated quantitative criteria were packet size, error percentage, and response time. The points gathered in each criterion are only valid for it.

IoT devices can send various variables to the middleware, which means that an object can send the humidity, temperature, and voltage. [16] prioritizes the Inatel Smart Campus scenario, in which most objects send 15 variables. Therefore, 1 sent variable weighted 0.3, 15 variables weights 0.6, and 100 variables weights 0.1. The Inatel Smart Campus is a real-life testbed where concepts and technologies for the Internet of Things can be validated. This project promotes innovation through cooperation between scientific research and corporations where products are evaluated, demonstrated and validated in a network that converges all types of IoT technologies and services. The Smart campus is located at INATEL (National Institute of Telecommunications), in Santa Rita do Sapucaí, MG – Brazil.

The study by Cruz *et al.* will be the basis for most of the performance evaluation studies through this thesis and concluded the best middleware solution depends on the chosen scenario. In scenarios with low number of devices, middleware solutions will perform similarly and the difference starts to be noticeable with more than 1000 concurrent users. Overall, Sitewhere presented better performance and stability throught the experiments, followed by Orion.

3.3 Performance evaluation through multi-criteria decision making

The main issue with most performance evaluation studies is that they merely determine the best solution for each criterion (category). Even with a detailed scoring system such as the one proposed in [16] (that will be the basis for the comparison using PROMETHEE), makes it difficult to determine the best global solution (that considers all the criteria) because comparing attributes of different nature is not intuitive. Therefore, people resort to determining the best in each category. To overcome such a problem, additional resources are needed. A resource that has proven to be effective is multiple-criteria decision making (MCDM). MCDMs are a field of operations research that can be applied to daily life aspects, such as business, industry, and technology. There are several MCDM methods to choose from, and all consist of establishing the evaluation criteria, attributing weight to each criterion, and establishing the same scoring system through the criteria (i.e., one criterion cannot have a maximum score of 10, while another can only gather a maximum of 9). In theory, unless the weighting system changes, the MCDM method's results will be identical regardless of the selected method.

3.3.1 Preference Ranking Organization Method for Enrichment Evaluation (PROMETHEE)

To rank alternatives in the best to worst order, several criteria should be evaluated. Moreover, these criteria are conflicting (i.e., usability vs. security). The literature offers multicriteria decision making (MCDM) approaches to overcome such issue [145]. One of these approaches is preference ranking organization method for enrichment evaluation (PROMETHEE), proposed by Brans and Vincke [146]. PROMETHEE is an MCDM outranking method to evaluate conflicting criteria using a finite set of alternatives, whose popularity is increasing based on the number of published papers in the literature [145]. This increase in popularity is because PROMETHEE is relatively simple to use in comparison to other MCDM approaches. Besides determining the best solution considering all criteria, PROMETHEE also allows qualitative metrics to be compared alongside quantitative metrics. The disadvantage of PROMETHEE and other MCDM methods is that it is impossible to eliminate subjectivity because some aspects will be prioritized through weighting.

PROMETHEE has more than one option (from I to VI) to apply. PROMETHEE II is the chosen tool for a complete ranking of alternatives in a finite set. It can offer a final list of alternatives, ranked from the best to the worst, based on the defined criteria and weights assigned to them. It compares criteria, pair by pair, starting by calculating the deviation (the difference between their values). Each criterion's difference must be applied to a preference function (there are six basic types to be chosen, depending on the kind of comparison is done). PROMETHEE II is based on five steps, as described in Algorithm I, considering [145] as follows:

- A a set of n alternatives to be compared in j different criteria ($j = 1, \dots, k$);
- $g_j(a)$ the evaluation (value) of alternative a ($a \in A$) in criterion j;
- $d_j(a, b)$ the difference between alternatives a and b ($a, b \in A$) in criterion j;
- $F_j[d_j(a, b)]$ the result of the preference function applied to $d_j(a, b)$;
- w_j the weight of criterion j.

Algorithm 1 – PROMETHEE II algorithm

	PROMETHEE II algorithm
1.	**Calculate the deviations between a pair of criteria (pair by pair)** $d_j(a,b) = g_j(a) - g_j(b)$ **(1)**
2.	**Apply the preference function** $P_j(a,b) = F_j[d_j(a,b)] \quad j = 1, \dots, k$ **(2)**
3.	**Calculate the global preference index (π)** $\forall a, b \in A, \quad \pi(a,b) = \sum_{j=1}^{k} w_j P_j(a,b)$ **(3)**
4.	**Calculate the outranking flows (positive $\emptyset^+$ and negative $\emptyset^-$), for each alternative** $\boldsymbol{\emptyset^+(a) = \frac{1}{n-1} \sum_{x \in A}^{k} \pi(a,x)}$ **(4)** **and** $\boldsymbol{\emptyset^-(a) = \frac{1}{n-1} \sum_{x \in A}^{k} \pi(x,a)}$ **(5)**
5.	**Calculate the net outranking flows (positive $\emptyset^+$ and negative $\emptyset^-$), for each alternative:** $\boldsymbol{\emptyset(a) = \emptyset^+(a) - \emptyset^-(a)}$ **(6)**

After all the steps, $\emptyset(a)$ for each alternative can be compared. The higher the value of $\emptyset(a)$, the better the alternative. In the end, the evaluation results in the ranking among alternatives, facilitating the choice for the decision-maker.

3.3.2 Performance evaluation through PROMETHEE

Comparing software is always tricky because they present functionalities that must be compared to other similar applications according to the organization's

requirements (through a qualitative comparison). However, if many solutions fulfill the requirements, a quantitative comparison is necessary. If more than one metric is available, users must decide which solution is the best in each category. Therefore, the most significant issue is defining the best global solution (considering all the requirements), which can be achieved through PROMETHEE II. The work present in [16] uses qualitative metrics to filter eleven solutions according to a given scenario. The five solutions that were compliant with the Inatel Smart Campus progressed to the quantitative comparison. For the MCDM methods to wor, the same scoring system must be applied through all the criteria. The solutions that will be evaluated using PROMETHEE II are Orion, Konker, Linksmart HDS, Sitewhere, and InatelPlat. More details regarding the considered software version can be found in [16].

The qualitative metrics used in this chapter are the communication methods with the server, the number of releases in 2017, and four security aspects. These security aspects include (a) Authentication per device, (b) different keys to send and consult data, (c) device-specific restrictions, and (d) IP address and MAC storage. The most popular application layer protocols for IoT are REST (Representational State Transfer), MQTT (Message Queueing Transport Protocol), and CoAP (Constrained Application Protocol). The Inatel smart campus scenario prioritizes REST communications (although REST is not efficient for most IoT applications). For this reason, solutions that support REST communications will gather 3 points, while MQTT and CoAP gather 1 point. There are other application protocols for IoT. However, they will not be scored because they are not as popular as MQTT and CoAP. For each security feature, a solution will gather 1,25 points. For each release, solutions will gather 1 point (the maximum is 5). Table 2 summarizes how each qualitative criterion were scored. Figure 7 presents the results of the qualitative comparison that is derived from the work of Cruz *et al.* [16].

Table 2 – *Qualitative Metrics Summarized.*

Metric	Scoring system description (maximum of 5 points for each criterion)
Application-layer protocols	Rest = 3 points CoAP = 1 point MQTT = 1 point
Security features	1,25 points for each security features
Number of releases	1 point for each release in the year (maximum of 5)

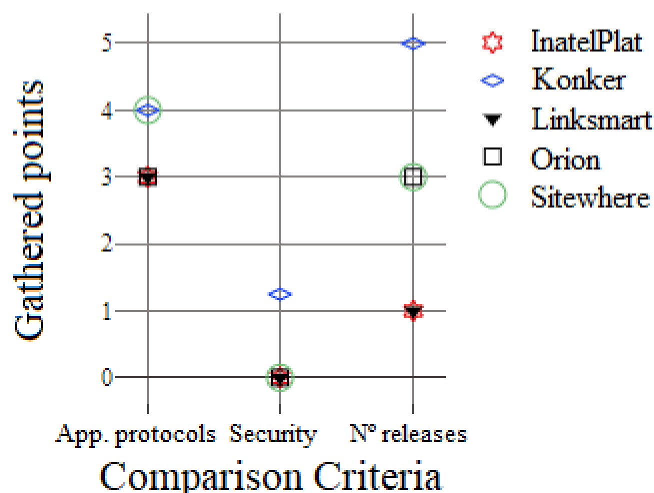

Figure 7 – *Results of the qualitative comparison considering application-layer protocols, security features, and number of releases.*

The used quantitative metrics are the following: packet size, error percentage, and response times with 1,000, 5,000, and 10,000 concurrent users. In each criterion, devices can send 1, 15, or 100 variables. Most devices at Inatel Smart Campus send 15 variables. It is unlikely that IoT devices will send more than 15 variables, so the experiment with 100 variables was performed to verify how the solutions deal with larger packets. The assigned weights were 0.6, 0.3, and 0.1 for 15, 1, and 100 variables respectively. The solution that presented the best performance gathered 5 points, the second-best 4, and so on. This is the same scoring system used in [16]. Table 3 summarizes how each quantitative criterion was scored.

Table 3 – *Quantitative Metrics Summarized.*

Metric	Scoring system description (maximum of 5 points for each criterion)		
	1 variable	**15 variables**	**100 variables**
Response times	0.3 – (5 to 1)	0.6 – (5 to 1)	0.1 – (5 to 1)
Packet size			
Error percentage			

Most real IoT environments will have a plethora of concurrent users. Therefore, after the initial scoring, experiments with more users were prioritized, and a weight of 0.7 was assigned to 10,000 concurrent users. Also, 5,000 and 1,000 users were assigned a weight of 0.2 and 0.1, respectively. Figure 8 summarizes the results of the quantitative comparison that is derived from the work of Cruz *et al.* [16].

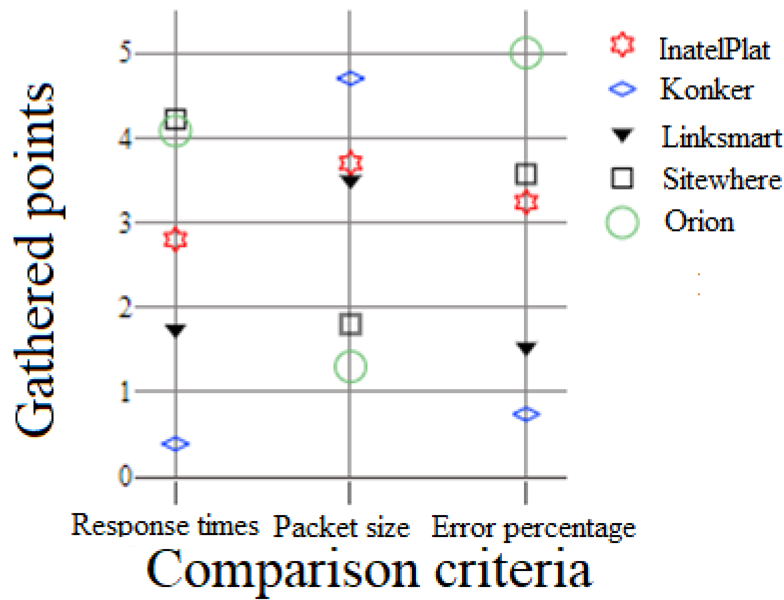

Figure 8 – *Results of the quantitative comparison considering response time, packet size, and error percentage.*

Overall, Konker performed better than its competitors regarding the qualitative criteria, followed by Sitewhere, while Linksmart and InatelPlat were at the opposite end. Furthermore, none of the solutions supported CoAP, and Konker was the only solution that offered per-device authentication. Regarding the quantitative criteria, Orion and Sitewhere performed better than its competitors. The scoring system emphasizes the difference between Sitewhere and Orion regarding the percentage of packet loss when, in reality, it is minimal. Despite being the smallest packet size solution, Konker performed poorly regarding response times and error percentage. InatelPlat was consistent across all the experiments, being the second-best scored regarding the packet size and third in the other criteria.

For the final comparison that will consider PROMETHEE, the qualitative metrics will represent 0.1 of the total weight, and all the criteria were attributed the same weight. The quantitative metrics represented 0.9 of the total weight, and the weight of each criterion changed according to five distinct scenarios, as follows: *i)* Same weight to all quantitative metrics (0.3); *ii)* Response times and packet size will be prioritized with the same weight (0.4); *iii)* Response times and error percentage will be prioritized with the same weight (0.4); *iv)* Packet size and error percentage prioritized with the same weight (0.4); and *v)* Response time prioritized (0.45), followed by packet size

(0.3). The results of the comparison through PROMETHEE are displayed in Figure 9 and originate from the results displayed in Figures 7 and 8.

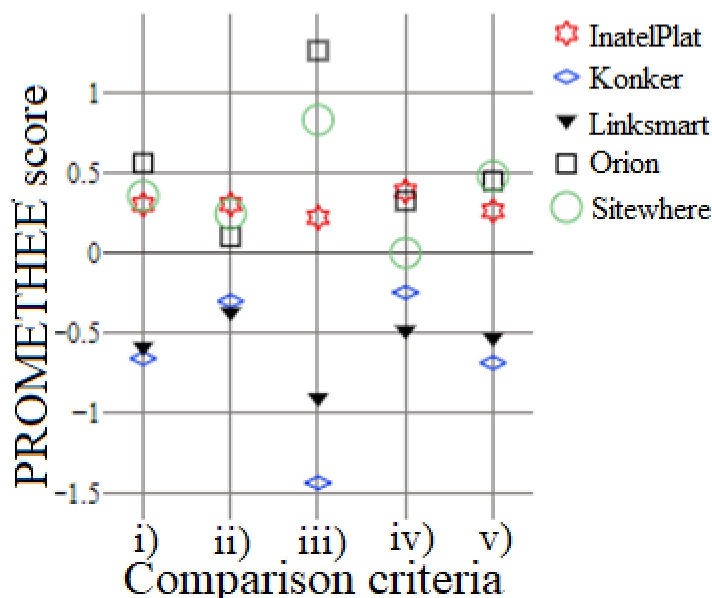

Figure 9 – *Results of the global comparison summarized for the five studied scenarios*

The results presented in Figure 3 demonstrate that the best solution depends on which criterium is prioritized and the only consistency in all the scenarios was Linksmart and Konker in the bottom positions. Also, Orion, InatelPlat, and Sitewhere presented the best performance in, at least, one of the proposed scenarios. In [16], Sitewhere was declared the best solution because its difference in comparison with Orion was minimal. The only criterion in which Orion was superior was error percentage (with a minimal difference). Therefore, scenario *v*) validates such a conclusion with Sitewhere performing better. In scenario *i*), Orion was the best solution, followed by Sitewhere. InatelPlat was the best solution in the scenario *ii*) followed by Sitewhere. In the scenario *iii*) Orion was the best solution followed by Sitewhere. InatelPlat was the best solution in the scenario *iv*) followed by Orion.

3.4 Summary

The results presented in this chapter confirmed that the best IoT middleware solution depends on which criteria are prioritized for a given scenario. Moreover, it confirms that MCDM methods can be used to determine the best solution for the chosen scenario, considering all the comparison criteria and are well adjusted for this type of problem. Such conclusions were possible by analyzing five distinct scenarios where different criteria were prioritized through weighting. Orion (a Fiware project), InatelPlat, and Sitewhere were considered the best solution in at least one of the proposed scenarios and performed better in the study, while Konker and Linksmart are at the opposite side (lower performance). Orion was the best solution in the scenarios *i*) (same weight to all quantitative metrics) and *iii*) (Response times and error percentage prioritized with the same weight). InatelPlat was the best in the scenarios *ii*) (Response times and packet size prioritized with the same weight) and *iv*) (Packet size and error percentage prioritized with the same weight). Sitewhere was the best solution in scenario *v*) (Response time prioritized, followed by packet size).

The study also demonstrated that neither of the solutions performs better across all scenarios. The reasons should be investigated because it could be related to the underlying programming language used to build the solutions, database management system, frameworks, or even security aspects. For this reason, building identical solutions with the same architecture using the most popular programming languages and comparing their performance could provide valuable insights.

4 Bridging Application Layer Protocols to HTTP

Internet of Things (IoT) environments are composed of countless objects produced by a variety of brands. Most IoT brands are only compatible with products from the same brand in the current status of IoT [147][148]. This restriction is inconvenient for the users because it can determine which brands will be acquired by users without losing functionalities [148]. When purchasing home appliances, such as a television or a microwave, users should know that compatibility will not be an issue. Furthermore, it is unlikely that one brand will produce every type of IoT device. Users will be limited to sets of brands or will purchase incompatible equipment without using all their functionalities if this issue is not addressed [149].

IoT devices' heterogeneity and compatibility may be easily solved through a worldwide standard [150][15]. However, standards are difficult to enforce [150], taking cellphone chargers as an example. In 2009, the European Union announced an agreement with ten (10) of the top cell phone manufacturers to adopt micro-USB as the connector to charge smartphones and tablets [151]. Such a standard would ensure fewer cables in landfills and was expected to be enforced by 2011. Unfortunately, some of the companies that agreed to such a treaty are yet to adhere to it.

In IoT, an alternative for providing interoperability support may be performing communication among nodes through an IoT middleware. An IoT middleware is a software that not only allows incompatible devices and applications to communicate but also stores the collected data for posterior treatment and analysis [15][152]. A middleware is deployed on a Web server, and data is transmitted from the device to the server through an application layer protocol. The most popular Internet application layer protocol is HTTP (Hypertext Transfer Protocol). However, it consumes too many resources [153], and alternative application layer protocols are being used for IoT communications [154]. Despite not being the “best” IoT protocol, HTTP is used in many IoT scenarios [155].

Unfortunately, most IoT middleware only supports two application layer protocols (HTTP and generally MQTT). If one of the user's gadgets is not compatible with application layer protocols supported by the middleware, such gadget will not achieve its potential regarding functionalities. In such cases, instead of sending a message straight to the middleware, an intermediary device translates the data into an application protocol supported by the middleware. The intermediate process would be seamless to senders and receivers. The thesis identifies the practical impact of such a solution and proposes MiddleBridge, an application layer gateway for IoT protocols that “translates” messages sent by Message Queuing Telemetry Transport Protocol (MQTT), Constrained Application Protocol (CoAP), Data Distribution Service (DDS), and WebSockets to HTTP.

4.1 Application layer gateways

Connected objects do not add much value by themselves, and remotely accessing objects' functionalities does not entirely justify the expected trillion USD impact. However, such devices' data can reveal users' behavior, which is extremely valuable because advertisers can target the groups that are more likely to purchase a particular service. Targeted advertisement represents a significant source of income for Facebook and Google [156], for example. The data collected in IoT environments are stored in an IoT middleware, a software that stores the collected data and allows incompatible devices and applications to communicate [16][141]. An IoT middleware is generally deployed in a robust server. Devices communicate with a middleware through an application layer protocol, and objects that are not compatible with the protocols supported by the middleware cannot be used to their full potential. An easy solution to overlap this issue consists of sending the data to an intermediary entity that translates the message into a protocol that is supported by a middleware. The software solution responsible for such adaptation is called Application Layer Bridge (ALB), Application Layer Gateway (ALG), or simply Bridge; the three terms can be used interchangeably in the literature. ALBs are similar to real-life translators, in which devices request the ALB to transmit a message in the desired protocol. A bridge operation is illustrated in Figure 10.

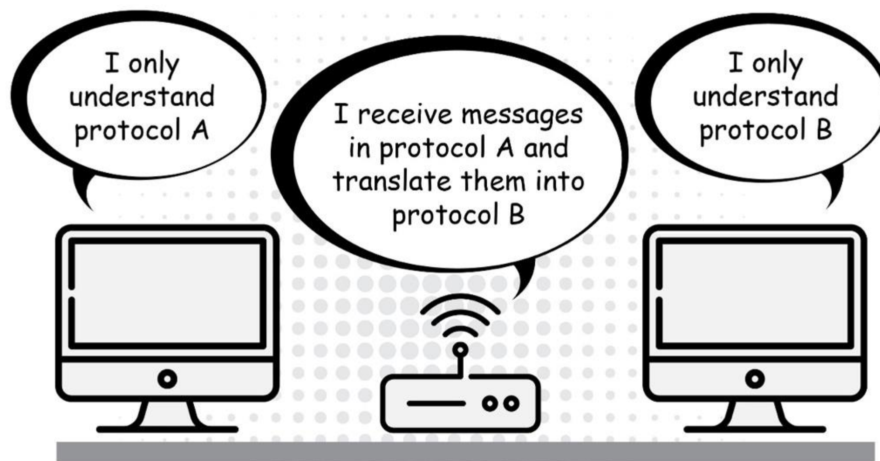

Figure 10 – *Illustration of a bridge operation where a gateway receives messages through protocol A and translates them to a protocol B.*

An application layer gateway is a software that intermediates the connection between an application server and the Internet. ALGs operation is straightforward since they receive a message through a protocol (protocol A) and forward the message in another protocol (protocol B). They are useful in IoT scenarios because most devices support a single application protocol (such as MQTT), and the server that stores the data might support a different protocol (such as HTTP). ALGs forward the received messages to the server, enabling seamless communication for users (the device and the platform do not notice the intermediary). This software allows incompatible devices to communicate with the middleware without changing the core code, neither in the middleware nor on the devices. In theory, ALGs can even be used to bypass restrictions imposed by a firewall or a network administrator where specific protocol traffic is blocked inside a network (i.e., BitTorrent).

All the ALGs follow a similar principle, and users can opt for every device to have a dedicated bridge (direct bridge) or share a single bridge for every device (indirect bridge) [157]. Regarding performance, a direct bridge is advantageous because only one device uses the bridge, and there is no concurrency for its resources. However, dedicating a gateway for every device is costly and increases the solution complexity. The concept of ALBs is not new in information and communication technologies as well as in IoT. Few ALGs have been proposed in the literature, such as Ponte [158] and

Gothings [159], both supporting MQTT, CoAP, and HTTP. Ponte is an Eclipse project and an evolution of the QEST broker [160].

Gothings and Ponte do not provide any functionalities beyond the “message translation”, merely sending a message with the same payload as an HTTP request in a different protocol. The proposed solution benefits from the smaller packet that is sent by IoT protocols and unlike other bridges in the literature, it does not receive full JSON or XML messages from the devices. Instead, MiddleBridge accepts a packet containing a smaller payload, reconstructs it, and forwards it to the IoT platform, which results in less stress for the constrained device. Additionally, the proposed solution provides a graphical user interface (GUI) that facilitates its deployment in IoT solutions managed by the average user (other bridges are configured through the command line).

4.2 Bridging from IoT Protocols to HTTP

MiddleBridge translates CoAP, Websockets, MQTT, and DDS requests into HTTP. It allows constrained devices to send less data when communicating with IoT middleware. The graphical user interface is simple and allows users to configure the conversion and forwarding of messages during runtime. MiddleBridge runs on any device with Java installed; this means that it can be deployed on a Raspberry Pi, personal computer, or even on the server that hosts the Middleware (in advanced cases where the user has access to the server). The solution includes embedded servers written in Java (for MQTT, it is called a broker).

The used MQTT broker is Moquette [161], RTI (Real-Time Innovations) connect DDS is used for DDS [162], the CoAP server is Californium [163], and the WebSockets deployment is TootallNate [164]. In all these protocols, the device sends data to MiddleBridge, which then forwards it to the Middleware chosen by the user (seamless to the sender and receiver). In the MQTT broker case, the proposed software subscribes to a topic specified by the user and then forwards the received data to the HTTP server. Furthermore, MiddleBridge is licensed under GNU GPL (General Public License) v3, which allows any third party to modify and distribute versions of the solution if they are open-source. The project is available at [165], and contains documentation explaining its usage. Figure 11 shows the MQTT GUI that was configured to send data to Orion context broker.

Figure 11 – *MiddleBridge's graphical user interface is configured to send data to the Orion context broker through the MQTT protocol.*

Most bridges follow a similar architecture because their operation method is simple and straightforward. MiddleBridge does not deviate from that basic architecture, and the only difference is the inclusion of a graphical user interface. Its architecture includes the following three elements: *i*) message translator, *ii*) protocol plugins, and *iii*) graphical user interface. Message translator is the entity responsible for receiving, modifying, and forwarding messages to the desired protocol. Protocol plugins are the implementation of specific application protocols themselves, while the communication among the different protocols must go through the message translator. The graphical user interface allows users to configure aspects related to the conversion and messages forwarding on runtime. On the device side, it must support at least one of the protocols supported by MiddleBridge. A minor modification is mandatory on the devices because they will send data to the proposed bridge instead of their default server. An intermediate understanding of the protocols that will be used for the requests is mandatory to apply the appropriate modifications to the IoT object. MiddleBridge documentation explains

the usage of each protocol, and it is available at [165]. If the gadget vendor does not provide the means to make such modifications, the bridging operation is not possible.

When devices send MQTT, CoAP, DDS, and Websocket messages, they will produce a packet with less overhead than a regular HTTP message. This means that if a device sends a payload A that contains a JSON message (i.e., {“name”:”NIA”}), it will always produce a packet size of at least 14 Bytes (1 Byte per character and the brackets also count) plus overhead. MiddleBridge takes such principle further and, instead of expecting a full JSON or XML message from the device in the example {“name”:”NIA”}, it merely expects a message containing “NIA”, which means that the device sends a packet with 3 Bytes from the payload plus overhead. The proposed software receives the message, converts it to an HTTP request, and forwards it to an IoT platform. Devices save time when sending data to MiddleBridge because, at the same time, it is forwarding a message to the IoT platform and ends the connection with the IoT device. MiddleBridge is an IoT device for the IoT platform and, for the IoT device, MiddleBridge is the IoT platform. This occurs because the process is seamless for the sender (device) and receiver (IoT platform). Figure 12 presents a flowchart of the MiddleBridge algorithm.

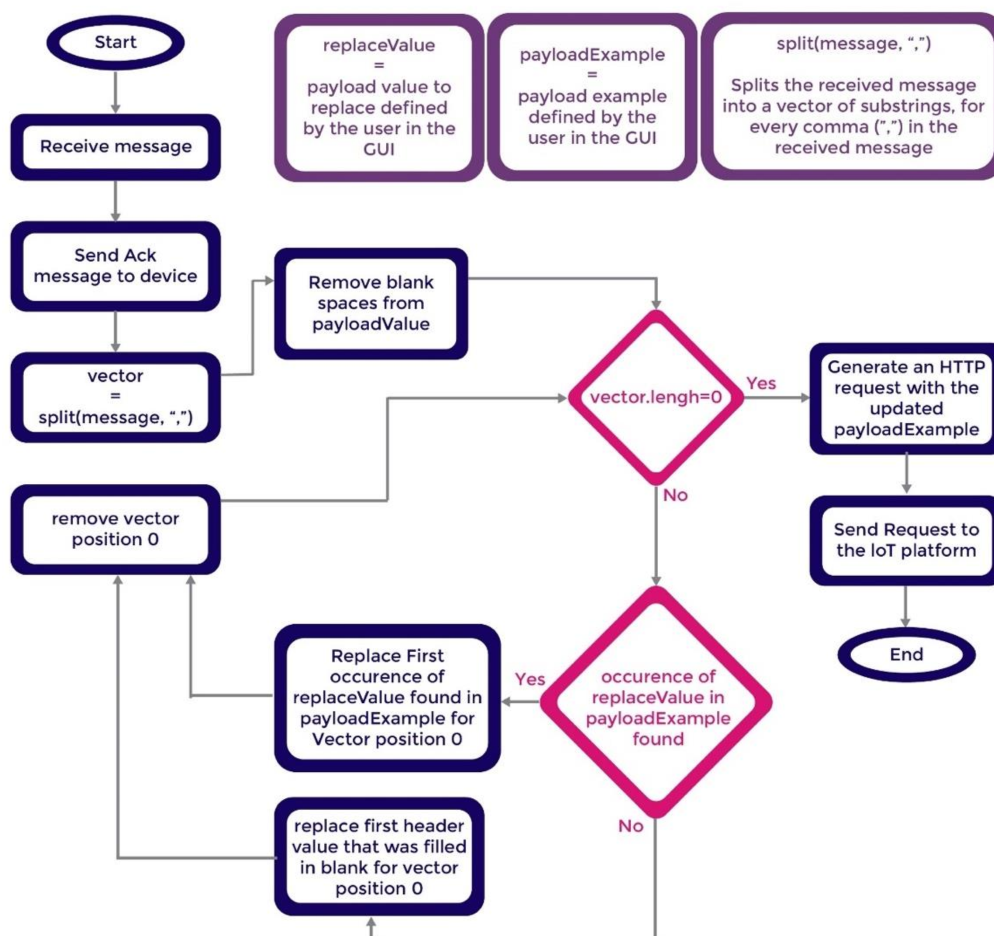

Figure 12 – *Flowchart explaining the MiddleBridge algorithm.*

To reduce the packet size sent by the IoT device, users specify the variables that will be sent (through the GUI). Therefore, instead of sending "Brightness:X", the device sends "X", a similar concept is applied to the headers, where the header's name and value are specified. If the header value is specific to each sender, users can specify the header name and leave the value blank. In the brightness example, the device sends 11 Bytes less on the payload. For each additional character on the payload, the message size is increased by 1 Byte (blank spaces also count [16]). When other bridges are used, the message contains the same payload as the HTTP request regardless of the used protocol. The reconstruction process is illustrated in Figure 13, and it is seamless to both sender and receiver.

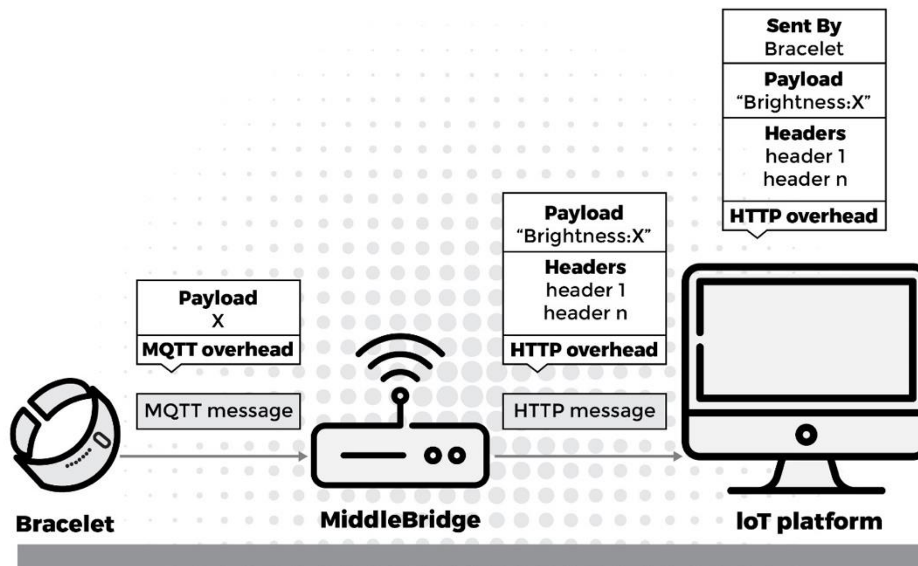

Figure 13 – *Illustration of a bracelet (i.e., an object) sending a small message to MiddleBridge that is reconstructed and sent to the IoT platform.*

Building this type of solution can be difficult because each supported protocol demands its server. Therefore, similar solutions may be difficult to run on computers with less than 256 MB of RAM (Random Access Memory). Moreover, almost every server of the supported protocols must be modified to process the received messages and forward them to the platform. The only exception to this rule are publish/subscribe protocols such as MQTT, in which software can subscribe to every message the server receives and forward it. However, finding an open-source deployment of the protocol server in the desired programming language can be challenging.

4.3 Orion context broker use case

Orion context broker [166] is a middleware platform developed by Fiware and follows a publish/subscribe deployment of the NGSI-9 and NGSI-10 Open RESTful API specifications. The data can only be sent to Orion through REST (Representational State Transfer). Therefore, Orion is an excellent candidate to evaluate MiddleBridge, namely because devices that support other IoT protocols such as CoAP, Websockets, or MQTT cannot send data directly to Orion. The solution's performance assessment regarding response times and packet size will be evaluated through experiments performed in a real wired LAN, and MiddleBridge was deployed in a Raspberry Pi 3. The Orion Context Broker was deployed on a Dell Precision 5820. The packets were

generated through Apache JMeter, which was deployed on another Dell Precision 5820 machine. Since Jmeter does not natively support MQTT, Websockets, or CoAP, the following community plugins were used [167], [168], and [169] for MQTT, Websockets, and CoAP, respectively. Although MiddleBridge supports the DDS protocol, it will not be evaluated in this thesis because there are no JMeter plugins for the DDS protocol.

Jmeter was also used to determine the packet size as well as the response times. The headers and body sent in an HTTP request are shown in Figure 2. Instead of sending a payload identical to it, the device sends a message through one of the supported protocols with the payload like "Luminaire,Luminaire1,99". The bridge application then reconstructs the rest of the message before sending the HTTP request, eliminating blank spaces beforehand. Since the goal of MiddleBridge is to reduce the stress in an IoT device, only the message sent by the IoT device to MiddleBridge was accounted for. When MiddleBridge receives the message, the previous communication with the device is closed. Therefore, the IoT device that sends the message is unaware of the remaining process (MiddleBridge reconstructing the message and forwarding it to an IoT platform). If the entire process is considered, the total network consumption and time for the packet to be delivered to the IoT platform will always be higher than a direct HTTP request (the same principle applies to response time). The hardware used in the experiments is shown in Figure 14.

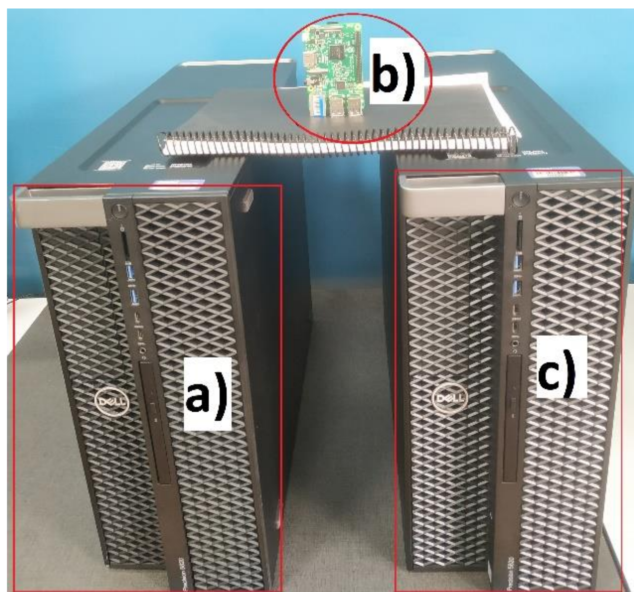

Figure 14 – *Photo of the hardware used in the experiments: a) Dell Precision 5820 that hosts Orion Context Broker b) Raspberry Pi 3 that hosts MiddleBridge; c) Dell Precision 5820 that hosts Apache Jmeter, used to generate the CoAP, MQTT, Websockets, and direct HTTP requests.*

4.4 Packet Size

The packet sizes are important for IoT devices because, in theory, they imply shorter transmission times. When communicating over the Internet, the receiver confirms the reception through an ACK (acknowledgment) or NACK (negative acknowledgment). Therefore, the size of the response message will be included in the experiments. The packets were generated through Apache Jmeter. Figure 15 presents the packet size analysis for a single request with MQTT, CoAP, WebSockets, and direct HTTP communication with the server.

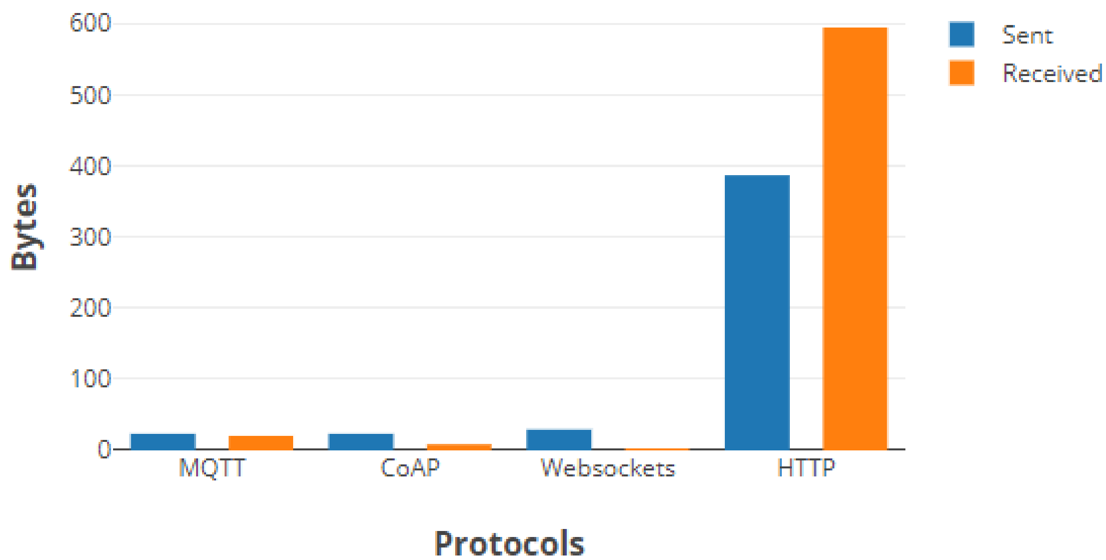

Figure 15 – *Analysis of the packet size (in bytes) of a single request with MQTT, CoAP, Websockets, and a direct HTTP Request to Orion Context Broker.*

It is observed that a direct HTTP request to Orion demands 401 sent Bytes and 595 received Bytes, and it is significantly less efficient than sending a request to MiddleBridge. Furthermore, there are no significant differences among the other three protocols. MQTT sends 23 and receives 20 Bytes, CoAP sends 23 and receive 8, while WebSockets send 29 and receive 2. Overall, using MiddleBridge, the sent packet size is 17 times smaller than a direct HTTP request and produces an ACK message 29, 74, and 297 times smaller with MQTT, CoAP, and Websockets, respectively.

4.5 Response times

Response times can be crucial in some IoT scenarios, such as healthcare, especially when the server is receiving several messages. When analyzing the response times, the round-trip time (RTT) to MiddleBridge for the different protocols will be evaluated. To verify the RTT, data were sent using a scenario with 100 concurrent users generated through Apache Jmeter. Figure 16 presents the analysis of the average response time with MQTT, CoAP, WebSockets, and direct HTTP communication with a server. All the requests were successfully delivered.

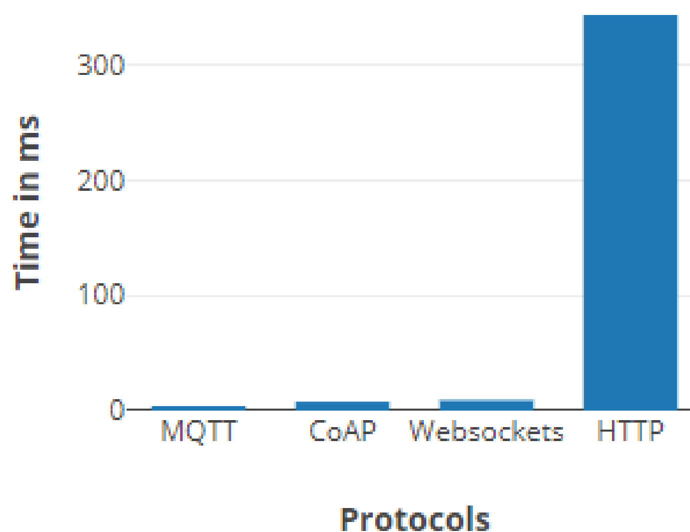

Figure 16 – *Analysis of the average response time (in milliseconds) with MQTT, CoAP, and WebSockets, and direct HTTP communication with a server.*

It is observed that with 100 concurrent users, a direct HTTP request to Orion demands on average 343.265ms, and it is inefficient when compared to a request to MiddleBridge. Also, there are no significant differences among the other three protocols, with MQTT averaging 2.101ms, CoAP 7.952ms, and Websockets 9.081ms. Overall, by sending their data to MiddleBridge, devices spend on average 163, 43, and 37 less time transmitting with MQTT, CoAP, and Websockets, respectively, than a direct HTTP request. The Orion Context Broker was deployed on the local LAN. In most IoT scenarios, the Middleware is located somewhere in the cloud, increasing the latency of a direct HTTP request, further increasing the efficiency of MiddleBridge.

4.6 Summary

The chapter proposed MiddleBridge, an IoT application layer gateway that converts MQTT, CoAP, Websockets, and DDS messages into HTTP. MiddleBridge can be deployed on any computer that runs a JVM (Java virtual machine) and allows IoT gadgets to seamlessly communicate with a Middleware. Unlike other application layer gateways, MiddleBridge can be configured through a graphical user interface and reduces the size of packets sent by an IoT device because a shorter message is sent to the bridge application that reconstructs it forwards to the middleware. The proposed solution is easy to deploy because all the servers of the supported protocols were embedded in the application. Moreover, the chapter assessed and demonstrated the efficiency of such a technique by evaluating the packet size and response times in a scenario where data is sent to Orion context broker (that only supports HTTP). The experiments were performed through Apache Jmeter. In the proposed scenario, packets that were transmitted to MiddleBridge were 17 times smaller than a direct HTTP request with any of the supported protocols. Also, with MiddleBridge, devices spent on average 163, 43, and 37 less time transmitting than with MQTT, CoAP, and Websockets, respectively. Furthermore, in most IoT scenarios, the IoT platform is located somewhere in the cloud, increasing latency, further improving the efficiency of MiddleBridge.

MiddleBridge is efficient because MQTT, CoAP, DDS, and Websocket messages produce less overhead than a regular HTTP message. Other bridges merely translate messages between the protocols, which is the only reduction in packet size they offer. The proposed software takes advantage of such principle and instead of expecting a full JSON or XML messages from the device like other bridges in the literature, it receives a message containing a smaller payload. Reconstructing it and forwarding it to the IoT platform resulting in less stress for the constrained IoT device. For the IoT platform, MiddleBridge is the IoT device, and for the IoT device, MiddleBridge is the IoT platform. Suppose the entire process is considered (device sends a message to MiddleBridge, which sends a response to the device, then processes and forwards the message to the IoT platform). The packet's total network consumption to be delivered to the IoT platform will always be higher than a direct HTTP request (the same principle applies to response time).

5 Performance Comparison of Programming Languages for Internet of Things Middleware

The sheer amount of connected devices has risen sharply in recent years. IoT devices have become cheap, easy to develop, and powerful. This allowed for the usage of connected microcontrollers in objects on an unprecedented scale. Such objects generate massive amounts of data and employ connection standards that differ from one another. This makes the creation of projects that use different types of devices complicated. While most systems use some form of RESTful APIs, many use messaging protocols (e.g., MQTT) or some other way of communication.

The IoT Middleware is a layer of abstraction between the devices and the user, managing different protocols and providing easy, unified access for the developer to its devices [141][142][152]. This IoT Middleware can be written on any of the existing programming languages. However, no experiments have been made to determine which programming language has the best performance for the needs of an IoT infrastructure, namely, a secure quick insertion of data with high availability that can easily scale [170][171].

To the best of the authors' knowledge, not many studies regarding programming languages performance comparisons exist, and most importantly, they usually focus on metrics like lines of code, memory usage, and execution times for algorithmic problems. Although those are important metrics for general-purpose studies, they are not the most important for a RESTful API, which is critical for IoT middleware. Usually, the first bottleneck is in the system's communication with external resources, such as databases or other APIs in a microservices architecture, and CPU/RAM usage merely a symptom of the mentioned bottleneck. For these reasons, Javascript's Node.js has become such a popular tool for backend development, Node.js' asynchronous nature [172] lets it execute code while, for example, it waits for a database response, thus not wasting precious processing time waiting for another service to complete its actions. Even though it is possible to create asynchronous applications in Java and Python, this

requires more work and significantly more experience from the programmer, while in Node.js, asynchronous behavior is the default.

This chapter aims to evaluate 3 programming languages (i.e. Java, Python, and Javascript) using the most relevant metrics for an IoT Middleware (number of simultaneous requests per second handled, response time, and failure rate) [173]. The chapter also provides insights for future works in the topic with a solid scientific base for choosing the most appropriate programming language when creating projects based on IoT infrastructures.

5.1 Previous performance evaluation studies regarding programming languages

IoT devices may adopt many different standards of communication, hence complicating the development of IoT applications. For this reason, an IoT Middleware is almost essential to a real-world IoT scenario, where a significant number of devices may be used in conjunction. Most middleware are generic and provide functionalities that can be applied to any scenario, such as data insertion and consultation. Although there are ad-hoc Middleware for industry and smart home applications, for example, which come with great UIs and field-related features.

Although some performance studies of Middleware solutions exist, they do not evaluate the effect of the underlying programming language in the solution's overall performance. Furthermore, the studies that compare programming languages generally focus on metrics such as lines of code, memory or CPU usage, and execution times for algorithmic problems. This approach provides some insight into the strengths and weaknesses of each programming language. Nonetheless, they do not capture the essence of what is important for IoT middleware, which is the communication with databases and the capability of handling simultaneous web requests [171].

Nanz et al. [174] made an extensive evaluation of 8 programming languages that representing the major programming paradigms (procedural: C [175] and Go [131]; object-oriented: C# [176] and Java [177]; functional: F# [178] and Haskell [179]; scripting: Python[180] and Ruby [181]). The study used coding problems from the website Rosetta Code [182], a large database of programming problems solved in many different programming languages. They evaluate a few different metrics, such as RAM

usage, error proneness, and code size, but the most important for this analysis is the code run time. The C language is the very clear winner, followed up by the Go language. However, they demonstrate that Python's performance is better than good enough for most modern problems, and more than that, Python can be faster than Java for simple and medium problems.

Prechelt [183] conducted a similar study involving Master's students and online submissions in 2000. However, his study was based on a single problem evaluation performed by many different programmers which included C, C++ [184], Java, Perl [185], Python, Rexx [186], and Tcl [187] programming languages. Nanz et al. [174], which ran a single solution per language for a great number of problems (over 700). He evaluated similar metrics in general as Nanz. Prechelt found similar results to Nanz: script languages (like Python) make for more concise programs but lack in speed and robustness when compared to compiled, strongly typed languages, like C. He also shows that, in general, Python is faster than Java, which is interesting, given the time passed between the studies and that different versions of the languages were used. This makes an even more solid case for Python's execution time's superiority.

Aruoba et al. [188] ran a comparison of the Stochastic Neoclassical Growth Model in many languages used often for economic analysis. The programming languages included in the study were C++, Fortran [189], Java, Julia [190], Python, Matlab [191], Mathematica [192], and R [193]. In their study, Java had a very significant advantage over Python, being 100x faster than the standard Python interpreter. The best result is, as expected, from C++, closely followed by Fortran. Javascript was not evaluated, neither by Prechelt nor Nanz et al. [174] [183].

Merelo et al. [194] used Scala [195], Lua [196], Perl, Javascript [197], Python, Go, Julia, C, C++, Java, and PHP [198] to solve common Genetic Algorithm operations. In their results, Java is the fastest overall, followed up by Go. Python and Javascript do a fine job as well. The interesting part of the experiments is that C++ was the worst of the batch in one particular case, which is quite surprising and very different from every other study. This could be due to implementation choices and optimizations in the other languages and possible libraries utilized.

Lei et al. [199] compared Node.js [200], PHP, and Python's performance, using ApacheBench [201] to simulate loads of 10.000 requests for different numbers of users. Results showed that Node.js is far better than PHP and Python in every situation tested,

averaging around 3.000 requests per second (RPS) and being more resilient to increased numbers of users. PHP had very good RPS (Requests Per Second) for under 100 users (around 2.500), but was very susceptible to large numbers of users, having worse performance than Python for over 500 users. Python had a modest, however robust, performance, averaging 500 RPS independently of the number of users.

5.2 Programming languages for REST APIs

There are currently many REST frameworks available in many different languages, although the most popular ones are written in Javascript, Java, and Python. PHP and Ruby also have very robust frameworks (e.g., Laravel and Rails). However, those are losing ground in recent years, mostly to Javascript's Node.js. In the meantime, newer languages like Dart and Go have delivered frameworks with very impressive performance. Knowledge of a framework's performance nowadays is mostly based on personal anecdotes, popular beliefs, or a few online tests [202][203]. Therefore, academic testing and benchmarking remain to be done on this subject.

The thesis evaluates 3 (three) programming languages, namely: Java, Javascript, and Python. The reason for choosing those languages was their popularity and the popularity of their frameworks. Based on Stack Overflow's Developer Survey 2019 [204], the most popular general-purpose languages (thus eliminating HTML/CSS and SQL) are Javascript, Python, and Java (see Figure 17).

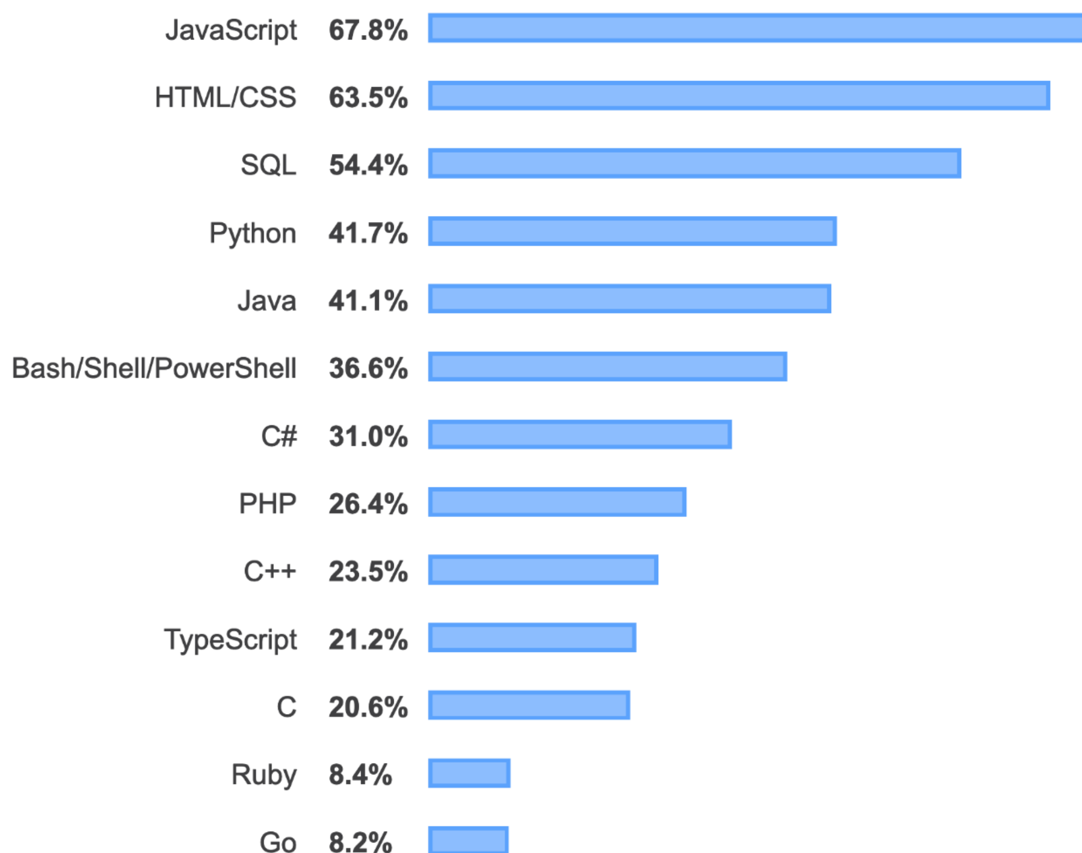

Figure 17 – *Most popular programming languages on Stack Overflow, 2019.*

Framework-wise, when eliminating Front-end Frameworks such as jQuery [205], React.js [206], Angular.js [207], and Vue.js [208], the remainder frameworks are: ASP.NET [209], Express [210], Spring [211], Django [212], Flask [213], Laravel [214], and Rails [215] as shown in Figure 18. Matching those with the three most popular programming languages the most popular frameworks are: Express (Javascript), Spring (Java), Django (Python), and Flask (Python).

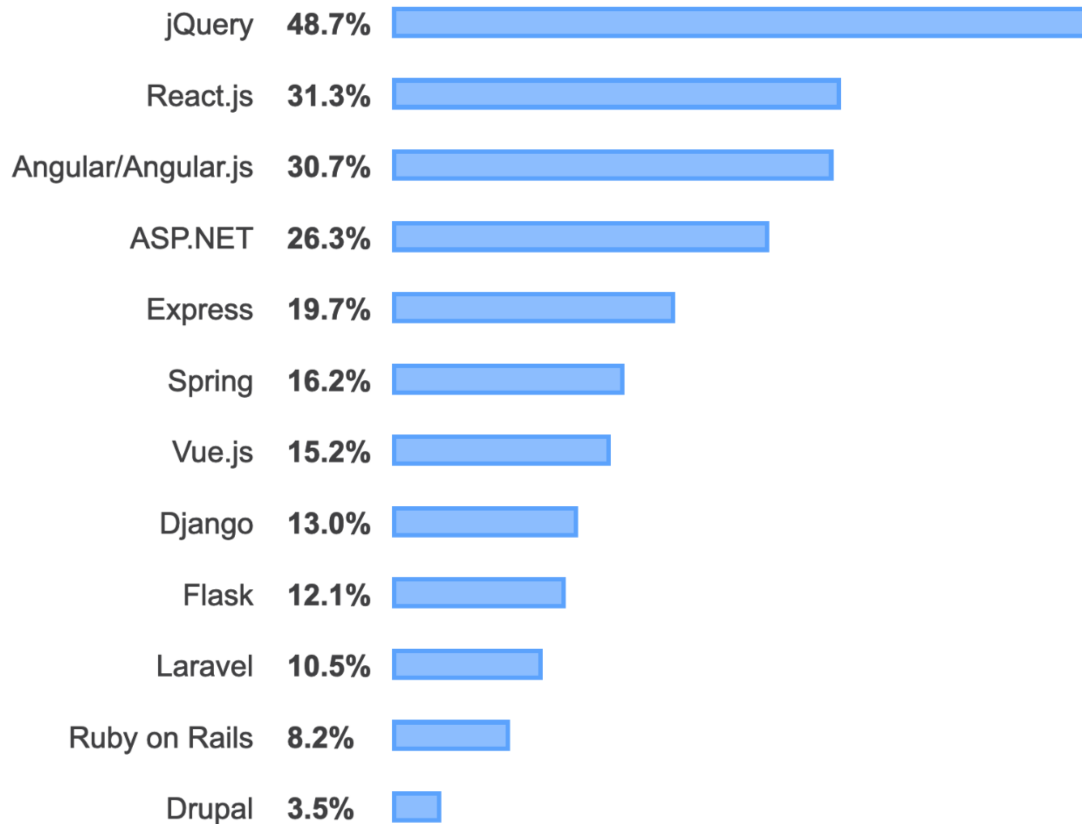

Figure 18 – *Most popular web frameworks on Stack Overflow, 2019.*

So the choice of Express and Spring is already obvious, since they were the most popular frameworks in their respective programming languages. However, the reason for choosing Flask over Django is that Django is a very feature-heavy framework, ideal for large web projects, while Flask is a light, microframework. Given the nature of the experiment, Flask was chosen over Django for its simplicity and avoided any unnecessary framework overhead.

5.3 Experimentation scenario

The proposed experiment involves load testing the APIs developed in different languages to measure the number of RPS achieved in each API, response times, and failure rates. For this, a tool called Locust was used to distribute the load in a cluster very easily using Kubernetes. The REST API created for this experiment is a simple POST route that follows the flowchart described in figure 19. Firstly, the API receives the POST request, which contains an authorization header, and the payload, which is a

JSON containing any number of parameters for insertion on the Database. The authorization header is a JSON Web Token (JWT) with the user information and an expiration timestamp. In the performed experiments, the requests are performed by Locust, an open-source software that is used as a testing tool to verify the scalability of solutions.

Once the request is received, the Middleware validates the authorization and, if successful, it proceeds to insert the data in its database, a MongoDB. After insertion, a response is sent to the device with only a status code and no body to reduce the size of the packet and make the transmission faster and more reliable. Everything is built inside a Docker container to facilitate the deployment of the different implementations. Each container has the API in one of the languages, MongoDB, and the required dependencies. The base image used is Ubuntu's official Docker image for version 16.04. The complete source code is available at Github [216].

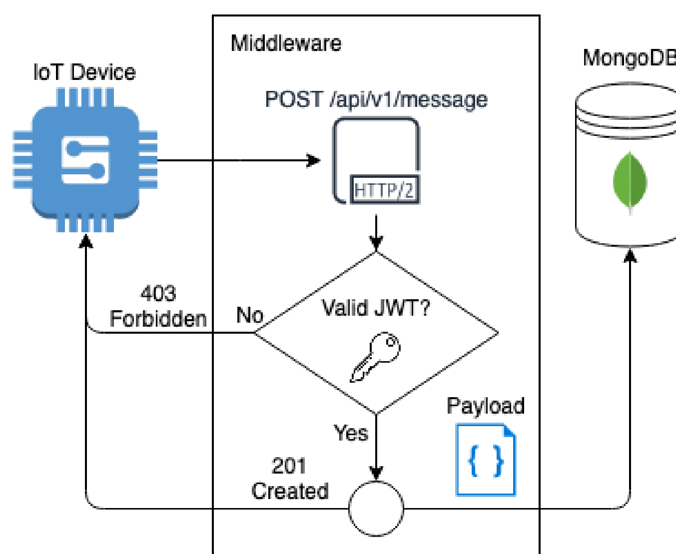

Figure 19 – *Data Insertion flowchart used in the experimentation scenario.*

JWT is used for validation because of its status as an industry standard, being a simple and safe, encrypted way of handling authentication between interested parties [217]. MongoDB was chosen as the Database because, as a NoSQL Database, it has 2 (two) important advantages for an IoT Middleware architecture: the fast insertion of data and the freedom to insert unstructured data. Those are very important characteristics, as the kinds of data that will be received are unknown to the Middleware. As the number of sensoring devices increase in the IoT environment, more data will be

transmitted. Therefore, more insertions into the database will be required. In that regard, MongoDB is one of the fastest Databases available today [218] [219] [220].

Although it is good practice in Docker to separate different applications in different containers, in this case, it was decided to deploy both the API and the Database in a single container to eliminate any lag in communication between containers. Other measures taken to reduce possible Docker overheads further were the usage of the host network, instead of the default bridge, and usage of volumes for each container. In these settings, Docker's overhead can be considered negligible [221]. The API container was deployed in an Amazon Web Services' (AWS) t3.xlarge instance, with 4 virtual CPUs, 16 GB of RAM, and an SSD.

Three APIs were developed for this test: one using Java's Spring Boot, one in Node.js' Express, and one in Python's Flask. The APIs were all kept as simple as possible, defining only the used route, importing only necessary libraries (such as MongoDB connectors), removing any logging statements after testing, and returning only the request's status code. Thus, any difference in results can only have three possible sources: the Framework used, the MongoDB connector (which differs between languages), and the base language itself. All of those are, in some way, related to the language, so the results should portray the different performances of the languages quite well.

Locust is an Open-source load testing tool made in Python [222]. There are many load testing tools available, the reason for choosing Locust was the ease of distributed deployment using Kubernetes [223], which enabled a much larger number of users than what could be achieved in a single machine. A Docker image was created, containing all the necessary dependencies to run Locust and a basic Locust script in Python, which has a class for making POST requests at the API being tested, changing only the number of parameters sent, that is configured by an environment variable. Locust ”min wait” and ”max wait” are both set to 1, which means each user will keep sending requests as soon as it receives a response from the server. Complete source code is available at Github [224].

Kubernetes is configured using 3 ”.yml” files, one for configuring Locust's master (the process that controls the workers/slaves*;* starts them and gathers the results), one for the slaves (the process that performs the requests and sends them to the master), and one for a load balancer service. Once deployed, Kubernetes will handle the pods' distribution and connection in an AWS cluster, created using AWS' EKS. The

cluster is set to create up to 100 AWS' EC2 type t2.xlarge instances, each with 4 (four) virtual CPUs and 16 (sixteen) GB of RAM. Kubernetes will assign one Locust master pod to any node, and one slave pod to all remaining vCPUs in the cluster. The load balancer service will allow communication to the master node. 400 (four hundred) slaves were used in total for each experiment, with 250 (two hundred and fifty) users per slave, creating a total of 100,000 (one hundred thousand) users.

The experiments were repeated for each API using 1 (one), 10 (ten), 100 (one hundred), and 1000 (one thousand) parameters per experiment. The experiment was run for 5 (five) minutes, and 100 (one hundred) seconds were dedicated for warm-up to allow locust to generate the requests. Since the experiments were performed on Amazon AWS servers, there were budget restrictions, and the authors of the thesis came to the conclusion that 5 (five) minutes was the minimum acceptable time to run the experiments. Such conclusion is derived from lesser scale experiments that were performed in local environments, that demonstrated that only occasional spikes (either low or hight) would occur), after 3 (three) minutes of experimentation. Thus, not altering the overall results of the experiment.

It is important to note that the slaves were running in multiple intances in the AWS cloud. Furthermore, each programming language had its dedicated server

5.4 Percentage of failed requests

The percentage of failed requests expresses the number of packets that did not achieve their destination or were not properly processed . Since most of the experiments were performed in the Amazon AWS environment and various instances produced the requests, it is safe to assume that failures were due to the programming languages. When it comes to failure rates, Python is particularly bad, delivering over 80% of failed requests for every number of parameters. In comparison, Java has around 13% of failure for under 1000 parameters, and Javascript under 10% of failures for under 1000 parameters. It is noteworthy that Javascript was much more affected by increasing numbers of parameters than the other languages, reaching almost 35% of failures for 1000 parameters, while Java reached 27.5%. Python maintained its enormous failure rates regardless of the number of parameters. Therefore, it is hard to determine if the

number of parameters makes a difference. The results are showcased in Figure 20, and all failures are due to the programming languages not being able to handle the high number of concurrent requests.

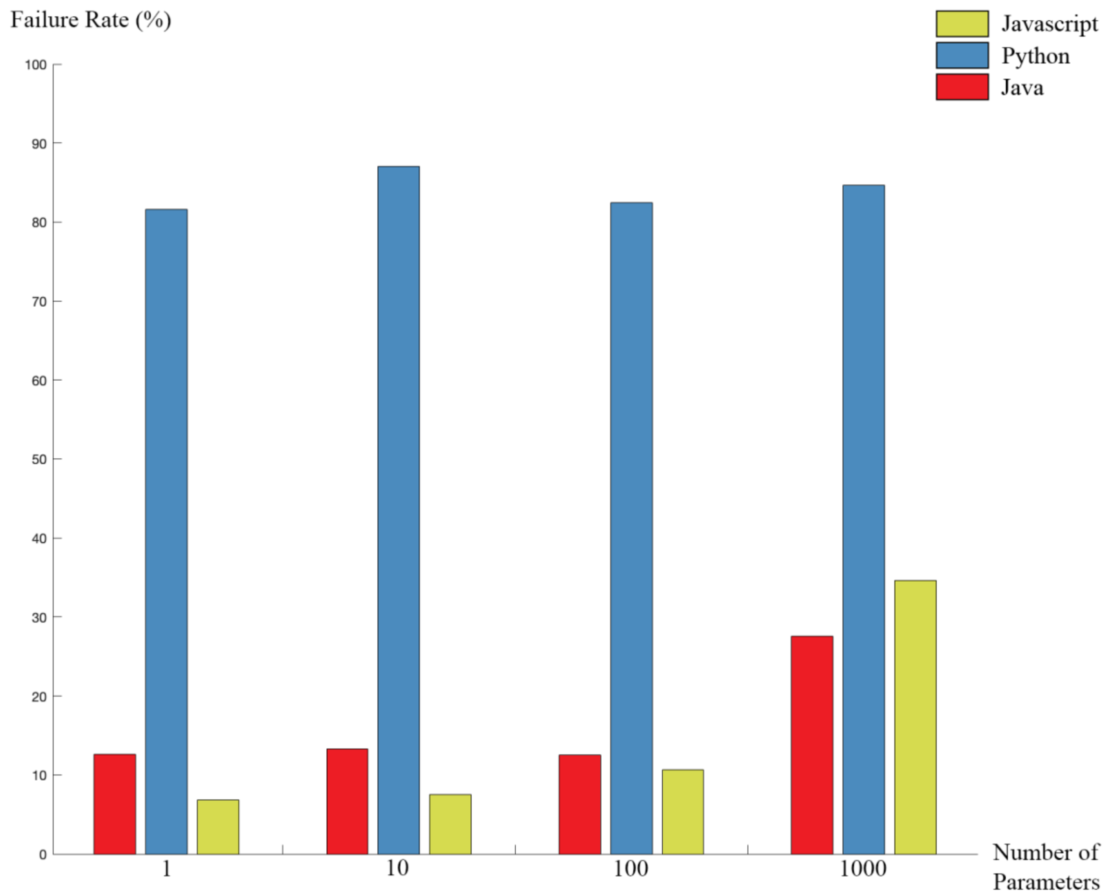

Figure 20 – *Percentage of failed requests by number of parameters.*

5.5 Requests per second

Results demonstrate that, for 1 (one) parameter, Javascript is the fastest language overall, averaging around 4,500 successful RPS, with a peak of 5700 RPS. In comparison, Java averages around 2600 RPS, after a warm-up period, peaking at 3300 RPS. Python averages 700 RPS and peaks at 800 RPS. Python's raw numbers approach 3,000 RPS. However, as shown in Figure 14, Python has over 80% of failures, meaning Python is refusing most connections to keep responding as much as possible. The results for 1 (one) parameter can be verified in Figure 21.

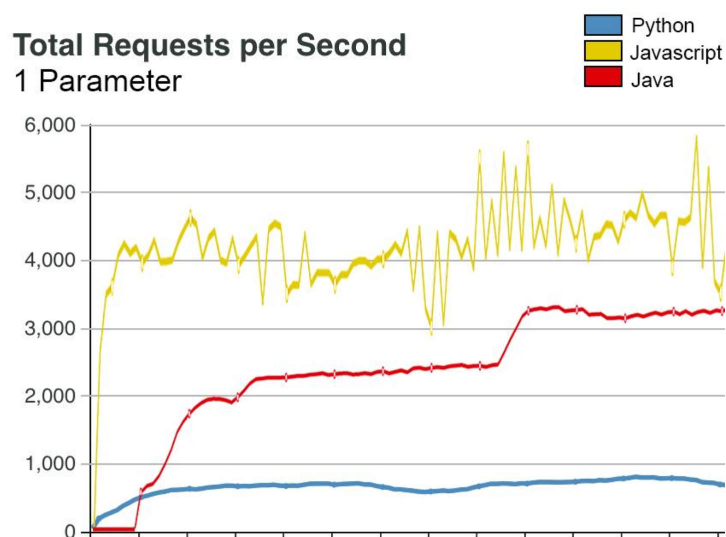

Figure 21 – *Successful requests per second for 1 parameter.*

For 10 (ten) parameters, Javascript and Java demonstrate similar behavior, with Java being slightly better, with an overall of 3800 successful RPS, peaking at 5100 RPS. Javascript, with an overall of 3600 successful RPS, peaking at 5150 RPS. Surprisingly, Python performs better with 10 (ten) parameters than 1 (one) parameter, with an overall of 2200 successful RPS, peaking at 3700 RPS. The results can be verified in Figure 22.

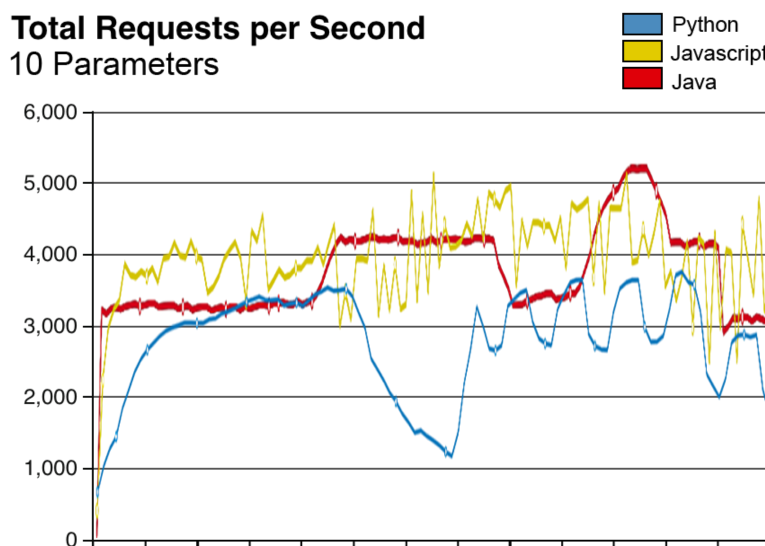

Figure 22 – *Successful requests per second for 10 parameters.*

For 100 (one hundred) parameters, Javascript and Java demonstrate similar behavior. Java, with an overall of 3000 successful RPS, peaking at 4800 RPS. Javascript, with an overall of 3000 successful RPS, peaking at 3800 RPS. Surprisingly, Python performs better with 100 (one hundred) parameters than 1 (one) parameter, with an overall of 2100 successful RPS, peaking at 3900 RPS. The results can be verified in Figure 23.

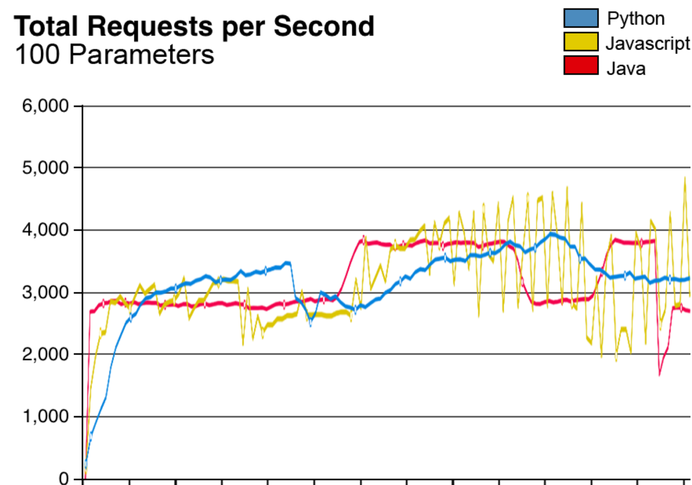

Figure 23 – *Successful requests per second for 100 parameters.*

For 1000 parameters, Java is clearly the most stable language, although Javascript can be faster at moments, its failure rate was higher, reaching almost 35%, while Java's is under 30%. The results can be analyzed on Figure 24.

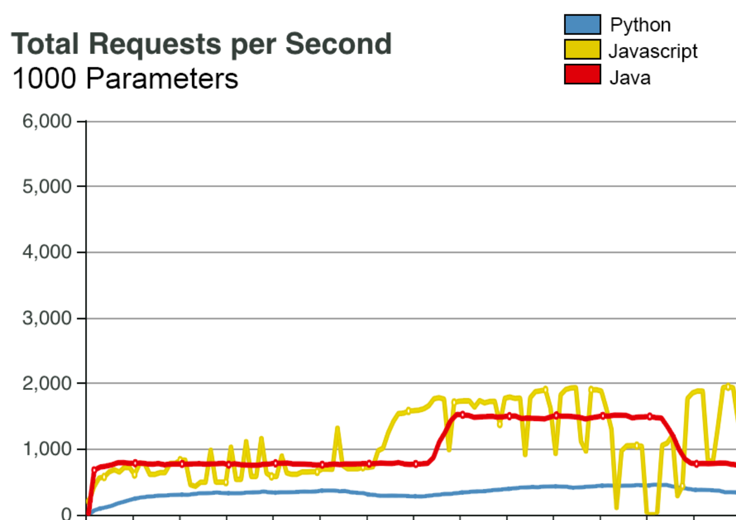

Figure 24 – *Successful requests per second for 1000 parameters.*

5.6 Summary

The chapter aimed to isolate the effects of the programming language on a REST API's performance. Results show that Javascript is the best language for up to 100 parameters. Java is the most robust overall, displaying similar behavior independently of the number of parameters. Python, despite its seemingly high throughput, has enormous failure rates and is not adequate for heavy loads.

Based on the observed results, it is safe to say that, for small to medium applications, if performance is the most important metric, than Javascript should be the language of choice. However, for data-heavy applications, or situations where robustness is the objective, Java is the best choice.

6 In.IoT – A New Middleware for Internet of Things

When building applications, the traditional approach considers dividing the application functionalities into small parts, called modules. This methodology is based on a modular programming paradigm, and each module contains the necessary tools to execute and accomplish a particular goal. When the application is fully built, all modules are generally combined into a single artifact, called a monolith. The modularization is mostly at the code level (e.g., the contents of the module are separated by folders). In practice, when a module needs a resource from a different module, it accesses it directly through the database, system files, or even memory.

Monoliths' issue lies in the fact that its modules cannot be independently executed, which means that an instance containing all modules must be deployed when scaling the application. Therefore, to create an instance of a highly requested module (i.e., the product module in an e-commerce website), an instance of the entire application must be deployed [225]. Furthermore, monoliths generally prevent the developers from using a programming language or a framework that differs from the original application [226]. Additionally, monoliths can be difficult to maintain because replacing or updating libraries can break an entire application instead of a single module.

6.1 Microservices overview

The issues presented by monoliths, especially in distributed systems, led to the wide adoption of the microservices architecture, which is viewed by some as an improved version of the Service Oriented Architecture (SOA) [227]. In computer science, a service is a group of software functionalities that execute a task for a client. Services are not necessarily tied to a specific application, and unlike real-life services, the service provider does not always require compensation for the provided service. A microservices architecture consists of developing an application composed of small

services that are not necessarily tied to a single programming language, are independently deployed, and communicate through network protocols (generally REST APIs or message queues) [228][229].

One of the most noticeable differences between microservices and SOA lies in communication among services in practical terms. SOA does not explicitly specify how services should communicate, while in microservices, it is clearly defined that they should communicate through network protocols. This difference means that when using SOA, two services that are deployed on the same physical machine can communicate by monitoring changes in a shared file, database, or even memory, which is not possible in a microservices architecture.

The rising popularity of microservices is driven by the acceptance of container technologies such as Docker, which are lighter and easier to deploy than virtual machines [230][231][232], and Kubernetes that manages containers [233][234]. Despite the microservices rising popularity and usefulness, monoliths are still broadly used because they consume less computational resources and are easy to maintain in small-scale applications. Therefore, microservices architecture should be used when scaling is needed, because it uses much more computational resources than a single monolith. Although all the microservices can be deployed in a single server, the architecture encourages distributing services among various servers.

6.2 The In.IoT architecture

In.IoT is an IoT middleware platform based on the reference architecture proposed by Cruz et al. [15] and represents a new concept for connecting things that are simple to deploy, use, and share. The name was chosen because a quick read reminds the phrase “in IoT”, and it is also easy to remember. The architectural requirements of In.IoT will be discussed as follows.

6.2.1 Data storage

A database is a collection of data that allows it to be easily stored, managed, and retrieved. To most users, the term “database” is synonymous with “relational database” and SQL (Structured Query Language). However, the lesser-known non-relational databases also exist and are increasing in acceptance with modern Internet applications'

growing demands. Although competitors, they serve different purposes, with NoSQL focusing on large volumes of unpredictable data [235], and SQL consisting of data integrity [236]. In practice, one of the most significant differences between NoSQL and SQL lies in the flexibility that is provided. When using SQL databases, all attributes of a given object must be clearly defined, and the relationship among objects is strictly enforced. With a NoSQL database, such constraints are not implemented, and any new field can be stored at any time.

Since NoSQL databases do not enforce the relationship among objects, they generally write and delete data faster than SQL, but are slower to retrieve data [237][238]. This flexibility provided by NoSQL databases facilitates a technique named sharding that consists of partitioning large databases into smaller parts [239]. With sharding, these partitions can be spread across multiple servers, increasing the scalability of the database, turning NoSQL databases useful for big data and IoT applications. For this reason, and based on the recommendation of a reference architecture proposed in [15], NoSQL databases will be used in In.IoT.

Another important architectural decision regarding data management is the use of multi-tenant databases, which allow each application data to be isolated and independent. Since, in theory, an entire application database can be stored on a single server, the performance of sharing can be increased even further. The downside of multi-tenancy is that each new database demands a few MegaBytes (MB) of additional storage space. Finally, In.IoT recommends indexing the username and creation date of the collection in which the data sent by devices is stored. This ensures efficiency in data consultation without impacting the performance.

6.2.2 Microservices and backward compatibility

In.IoT compatible architectures should use the microservices architecture and possess a service discovery mechanism, a gateway service for each supported protocol, and a load balancer for each protocol. The service discovery mechanism is crucial for the scalability of the solution because it allows new services to register themselves, and other services to consume it. Companies such as Netflix rely on thousands of services, and a human user cannot manually register every service. The API gateway for each protocol guarantees that the final user consumes multiple instances of the same service without being aware of their existence, because the user accesses the service through

the same route or port. Furthermore, the API gateway should distribute incoming traffic among the instances of the service through a load balancer. Finally, the API should be versioned to guarantee backward compatibility, ensuring that older devices never lose functionalities. Before different microservices exchange data among them, they authenticate themselves in the Service discovery. Figure 25 summarizes the data storage as well as the secure communication among services after authentication on the In.IoT microservices architecture. All the microservices can run on a single server or distributed among various servers.

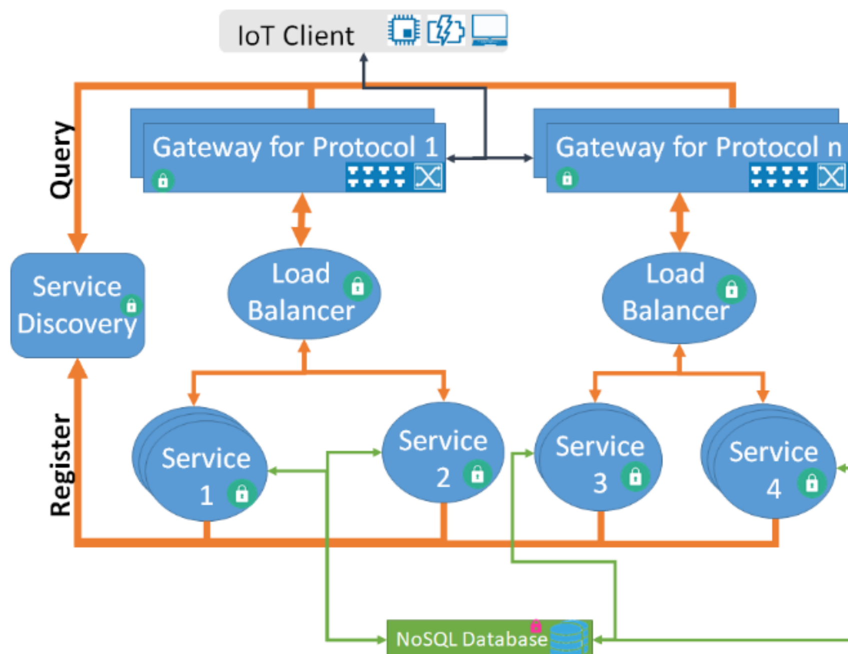

Figure 25 – *Illustration of In.IoT microservices architecture and data storage.*

6.2.3 Security

In.IoT architecture envisions security features such as multi-tenancy at the database level, which means that each application possesses a dedicated database. Another security feature is related to a clear distinction between human users (administrators) and device users, which guarantees that every device and administrator has individual credentials (username and password). Furthermore, all passwords should be encrypted to reduce the effects of a possible database breach. Additionally, the only protocol that directly accesses database should be HTTP through REST methods. This means that when a user authenticates in the MQTT broker service, the In.IoT MQTT service performs an HTTP request to another In.IoT service to validate the user identity. This approach ensures only one persistence method and guarantees that data will always

be stored/consulted in the same way, providing additional security and avoiding that data is processed differently over the various protocols.

JWTs (JSON Web Token) are used as the authorization mechanism because they are self-contained and signed by the server, which is good for performance. The tradeoff of using JWTs is that they increase the size of the message that is sent to the server. The specification for JWT states that it should start with the word “Bearer”. It is used to imply that the application/user requesting service may be the “bearer” of a previously agreed token. In practice, when a request is received, the first validation searches the word “Bearer”. In In.IoT deployment, the word “Bearer” is not used at the beginning of a token to optimize its performance. JWT is the mainstream deployment of an authorization mechanism, called client token, where the server generates a digitally signed token that can contain any user data, such as a username [136]. This token is secure because the user can only deceive the server if he possesses the password used by the server to digitally sign the token. This technique also allows the server to not store any information regarding the session.

The security of the JWT lies on the insurance that the user did not change its data [136]. Therefore, hiding data contained in the token is not one of JWTs goals, and such data is decoded without the secret. For this reason, only identifiers are encoded in the JWT, and sensitive information, such as passwords, are not placed inside the JWT. An issue with this technique is the tokens cannot be revoked without changing the secret, which would affect all the users. Another approach could consist of restricting users from generating new tokens or even implementing additional validation on the server-side before attending the request. One disadvantage of JWT is they require over 100 characters, which significantly increase the packet size.

To further increase security, In.IoT suggests some modifications to the MQTT broker to avoid users from “eavesdropping” unauthorized communications. The MQTT specification does not address which topics are accessible after users are connected, and users can publish and subscribe to multiple topics, and these modifications are crucial. To understand these restrictions and why they are crucial, it is essential to understand MQTT topics' syntax. The topic identifies where the data will be published or subscribed, and the topics possess levels, which are separated by a “/” (slash) character. Topic levels allow users to make sense of the topic's purpose; e.g., from the topic

“home/room1/bed/temperature”, users can infer that it is related to the bed's temperature located in room1.

Another concept related to MQTT topics is wildcards, which allow clients to subscribe to multiple topics simultaneously, without even knowing their existence. The “#” wildcard allows users to subscribe to every topic starting from the level it was placed. For example, by subscribing to the topic “#” (hashtag), users can access every communication on the broker. By subscribing to the topic “home/#”, users have access to every communication with the prefix “home/”. The “+” (plus) wildcard allows users to subscribe to the next level of a topic, which means that a user subscribed to “home/+/+/+” has access to everything that is published on the next three (3) sub-levels of the “home/” topic.

The first restriction imposed by In.IoT is not allowing the “#” wildcard, impeding users’ ability to subscribe to every communication on the broker. The second of these restrictions is allowing only one use of the “+” wildcard, to avoid that users subscribe to topics such as “+/+/+/+/+”, effectively reaching the same problem as the “#” wildcard. The third restriction is that when subscribing or publishing to topics, they should start with the application name that is generated when a human user registers and is inherited by all of its devices. The application name should be verified on each subscription/publication and negates users' possibility of accessing other application data. The fourth restriction consists of obligating users and devices to publish on topics that end with their username. This will prevent security critic devices from accepting unauthorized requests from rogue devices or users.

6.2.4 Improving overall performance

To increase the performance when persisting device events, In.IoT attempts to reduce the number of consultations performed to the database before persisting device events because accessing data on the RAM is much faster than accessing data on a Hard Drive. Assuming the operations that require access to a database are denoted to as A_D and memory operations are denoted to as A_M, a simple analysis can be performed. The traditional approach using any authorization mechanism includes querying a database if an user exists and is authorized to perform the requested actions (A_D). Then, the solutions verify if the request is well-formed (A_M), contains the required fields (A_M), and writes data into a database (A_D). After saving data, solutions start building the response message, which includes a reply code (A_M), reply headers (A_M), and a reply

body (A_M). Finally, the operation is logged in a terminal (A_M) or a text file (A_D). This results in 2 database accesses and 6 memory accesses ($2A_D + 6A_M$) in a scenario where an operation is logged only in a terminal.

In.IoT uses JWTs as the authorization mechanism and takes advantage of the fact that JWTs are self-contained, have an expiration date, and are signed by the server. In.IoT places the username and application information inside the JWT and trusts it, which avoids additional queries to the database. This is done because unauthorized users can only obtain a valid JWT if they intercept and decode a valid request or discovers the server's private key. In both cases, the additional validation would not prevent the attacker from sending data, and nothing can be done. Therefore, In.IoT approach validates JWT (A_M), since there are no required fields, it is verified if a message is well-formed (A_M), and data is written to a database (A_D). In.IoT response message only includes the reply code (A_M) and the minimum required headers (A_M). Finally, no reply body is included in the response message if the request is successful, and only errors are logged. This results in 1 (one) database access and 4 (four) memory accesses (A_D + $4A_M$). In.IoT efficiency can be verified in equations (1), (2), and (3). In all of the equations, In.IoT number of accesses is on the right inside of the equation. In equation (1), the logs are displayed on a terminal, (2) logs are persisted on a log file, and (3) logs are displayed on a terminal and persisted on a log file. The efficiency of this approach can be verified by the results of the experiments presented in Figure 26.

$$2\,A_D + 6\,A_M > A_D + 4\,A_M \qquad (1)$$

$$3\,A_D + 5\,A_M > A_D + 4\,A_M \qquad (2)$$

$$3\,A_D + 6\,A_M > A_D + 4\,A_M \qquad (3)$$

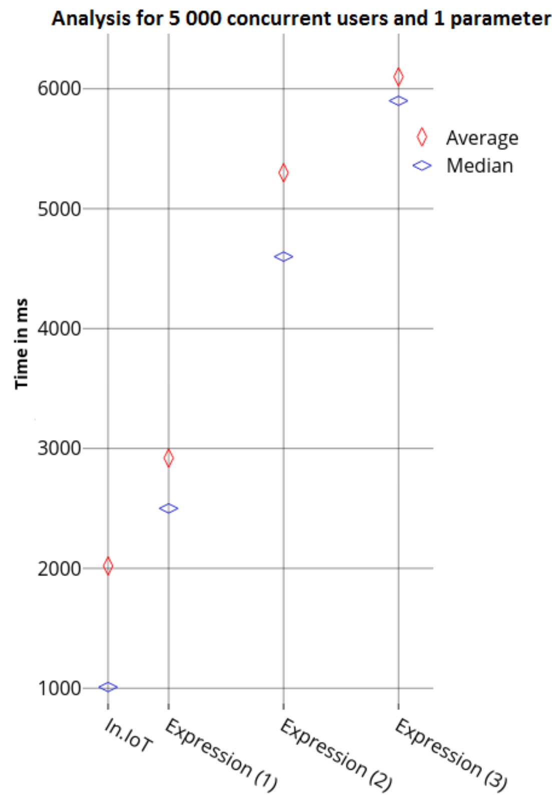

Figure 26 – *Response time analysis for 5 000 concurrent users where 1 parameter were sent considering In.IoT and the modifications presented in Expressions (1), (2), and (3).*

6.3 Construction of the In.IoT middleware

Chapter 5 concluded that for data-heavy applications where robustness is one of the key requirements, Java is the best choice. The only downside is that Java is a "heavy" programming language, but since an IoT middleware is intended to be placed on a powerful server, resources are not an issue. Java was also chosen because compatibility with any Operating System is a priority for In.IoT first deployment.

Since the microservices architecture is being used, other services that perform analytics can be written in a more appropriate programming language such as Python without impacting the core services. Additionally, In.IoT uses Netflix Eureka [240] as the service discovery mechanism, Netflix Zuul [241] as the API gateway, and Netflix Ribbon [242] as the load balancer. Spring Cloud Netflix was chosen for these core tasks because it is entirely open-source and is used by Netflix, proving its scalability [243][244].

The other services are customized versions of other open-source solutions. The MQTT broker used is a customized version of the Moquette MQTT broker built with Java [245]. The modifications made to Moquette were prepared to comply with In.IoT

architectural security demands and allows it to register new instances on Eureka. The repository can be found in [246]. The MQTT gateway service that is used is a modified version of the Raw TCP Proxy [247]. The MQTT gateway service can be found in [248]. The CoAP server is a customized version of the Californium CoAP server [249], and the CoAP gateway is based on the UDP Proxy duplicator [250]. The CoaP server is available at [251] and the CoAP gateway service can be found in [252].

The most notable modification made to the MQTT gateway, CoAP service, and CoAP gateway is Spring cloud support. Moreover, MongoDB was the chosen NoSQL database because of its popularity, performance, and compatibility with all the popular programming languages, including Java [253]. Full deployment of all the In.IoT services requires approximately 8 GB of RAM to run in a single server, and can be performed via an automated script. Each microservice can also be executed in an individual server (it is also encouraged by the Microservices architecture). As a general recommendation, the server administrator should follow the best practices and guidelines to secure the database as well as the server that hosts the middleware.

6.3.1 In.IoT features relative to the reference modules

The work of Cruz et al. [15] proposed a reference architecture for IoT middleware solutions. This architecture envisions a platform that is composed of six modules. It considers important requirements, such as scalability, reliability, event management, security, and resource discovery. The modules of this reference architecture are the following: *i*) Interoperability, *ii*) persistence and analytics, *iii*) context, *iv*) resource and event, *v*) security, and *vi*) Graphical User Interface (GUI).

In.IoT complies with **a middleware reference architecture's security module recommendations**, including individual device credentials and allowing authorized devices to access other device data. The recommendation refers that a device IP address should be stored whenever it sends data, detecting device habits, and alerting users that behavior changes is planned for future releases. The recommendation to use different credentials to publish and consult data in devices is not followed because there are more efficient ways to achieve this, such as public devices with surrogate usernames, which can be granted and revoked by an admin user at any time.

Regarding the **interoperability module**, In.IoT supports MQTT, CoAP, and HTTP at the application layer. SDKs written in Java and Arduino are provided, but other

SDKs will be provided in the future. These SDKs help developers to create software for a specific platform. For example, the Arduino SDK provides code that is ready to be deployed in an Arduino compatible device, enabling it to consult and publish data in In.IoT. Although these SDKs are offered, their usage is not mandatory.Concerning the **persistence and analytics module**, the only recommendation that is not followed by In.IoT is to process or feed the data to machine learning algorithms (or external sources that are capable) to reveal hidden patterns. However, this feature is planned and will be used for security purposes.

The recommendations of the **Resource and events module** are considered, and In.IoT allows devices to share their capabilities and how they can be used through a JSON message. This message can be sent when the device is created or by manually editing the device in the Administrator interface. This JSON message is displayed in Figure 27 (a). With such method, devices can query the MQTT broker for available services by posting "!v1find " followed by the intended ability and option, as well as the reply topic. The broker then sends a request to In.IoT API, and if a match is found, the JSON is sent to the device that requested the service, and is a self-explanatory manual.

(a)

```
1 {
2 "username":"defined@ByUser.com/autoRegister",
3 "password":"definedByUser",
4 "name": "coffee machine",
5 "abilities": ["coffee"],
6 "model": "fxcsqw",
7 "abilityDetails": {
8   "coffee": {
9     "type":"actuator",
10    "listenTopic": "/home/coffeeMaker/listen/+",
11    "replyTopic": "/home/coffeeMaker/reply/+",
12    "listenMessage": { "coffee": "#option" },
13    "replyMessage": {
14      "code": "#replyCode",
15      "message": "#replyCodeText"
16    },
17    "replyCode": {"success": 1, "waterFailure": 2 },
18    "replyText":{"success": "S", "waterFailure": "F"},
19    "options": { "java": 1, "cappuccino": 2 }
20  }
21 }
22 }
```

(b)

```
23 {
24   "Username":"c8fa25ca-14ff-4e2d-8813-f5396089727a",
25   "Password":"e#*X4tTMqdgP&1V"
26 }
```

Figure 27 – *Example of (a) an auto-register JSON from a coffee machine and (b) the unique credentials returned by In.IoT.*

In the future, if more middleware adopt this approach, devices such as Alexa who are constantly listening, can increase their utility. To the best of the authors' knowledge, In.IoT is the first middleware platform with such an approach, and this method will receive incremental improvements. In.IoT also supports Over the Air (OTA) updates, allowing devices to upgrade their firmware. In.IoT goes beyond the resource and events module's recommendations, allowing devices to securely register themselves in the platform through surrogate usernames and passwords defined by the user. These surrogate credentials possess an expiration time, and the devices can inherit some characteristics that were configured by the user. Figure 27 (a) shows a coffee machine presenting its capabilities to In.IoT using the auto-registration feature and (b) In.IoT sending the unique credentials. The JSON that a device would receive by entering "!v1find coffee !v1replyTo /home/device/+" on a topic is similar to the one presented in Figure 27 (a) minus the auto-registration details from lines 2 and 3. Note that all the messages can be encrypted by installing a security certificate in the server. In this thesis, messages are displayed in plain text just because, otherwise, readers would not be able to read and follow their logic.

Lines 2 and 3 are the auto-registration username and password that were configured by the user and transferred to a device, enabling it to auto-register. Lines 4 – 19 are configured by the device manufacturer and act as a self-explanatory manual. Line 5 presents the device capabilities, while lines 7 – 19 detail how they can be used. Line 10 is the topic where a device will be listening for queries, and line 11 states the topic where a device will reply in case of success of failure. Line 12 presents how the device should be queried (the options are presented in Line 19). The auto-register function is the In.IoT attempt to produce a new standard for devices to securely register themselves and share their abilities with minimal human interference. When devices send valid auto-registration credentials to In.IoT, it replies with the true credentials that should be saved and used by the device. An example of such a reply is presented in Figure 27 (b). This process should be done once in a device lifetime. As a proof of concept, a WiFi deployment of this concept will be demonstrated, but it can be replicated in any other technology. Figure 28 shows an IoT device that possesses a button and three LEDs. This device has three operation modes: *i*) receiver, *ii*) hotspot, and *iii*) configured. The LEDs act as a visual tip so users know in which mode the device is configured to operate, and the modes can be changed by pressing a button.

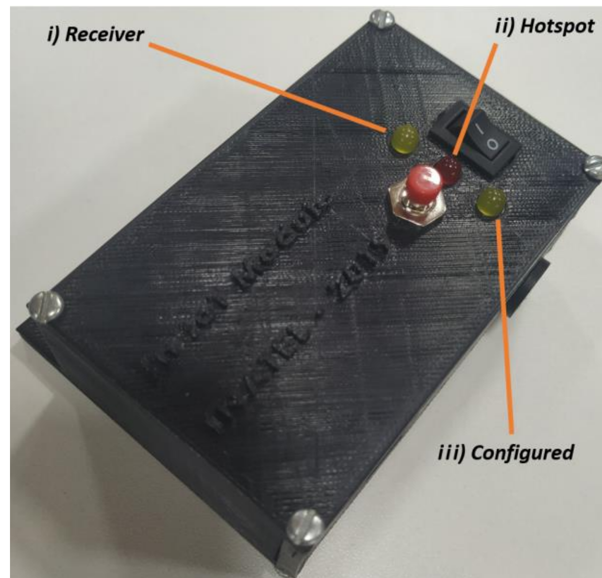

Figure 28 – *Example of how the auto-registration function can be used by a device with three operation modes: i) receiver, ii) hotspot, and iii) configured.*

When the **receiver mode** is active, the device queries for a WiFi network with a pre-defined SSID and password (this is setup by the device makers). The user then configures a WiFi hotspot with his smartphone; any modern device has this ability. Then, the user accesses a mobile App that is connected to In.IoT, logs in his account and introduces an auto-register username and password, as well as the SSID and password of his home WiFi network. To avoid duplicates, the device auto-register username is always preceded by the user e-mail. Once the IoT device is connected to the WiFi network configured by the user, it queries the network's first valid IP (which will always be the cellphone) for the auto-registration credentials. When a device receives a valid response, it uses this information to connect to the users' home network and then auto-register himself with In.IoT. After this, the device changes to **configured mode**. The receiver mode is useful to connect various devices at once

When the **hotspot mode** is active, the device creates a WiFi network with a pre-defined SSID and password (the device makers set this up). The user then connects to this network with his smartphone, opens the Web Browser, and accesses the first valid IP of the network (which will always be the IoT device). Then, manually introduces an auto-register username and password, as well as the SSID and password of his home WiFi network. Once the user presses "send", the IoT device receives this data and uses it to connect to the network, and auto-registers himself in In.IoT. After this, the device

changes to **configured mode**. The hotspot mode is useful to connect a single device in cases where the user cannot create an additional WiFi network.

After the device registers itself, the details of the new SSID network, as well as the credentials, should be encrypted and stored on non-volatile memory, so the device can access them after losing power. Regarding encryption, it should be done to prevent attackers with physical access to the device from obtaining the credentials.

6.4 Performance evaluation and demonstration

Since the microservices architecture consumes many computational resources, In.IoT can be deployed without the service discovery, and API gateway for the corresponding service to preserve resources. Additionally, if the MQTT protocol or CoAP protocol is not relevant for the solution, the corresponding services can also not be deployed. Therefore, In.IoT can even be deployed on “low-end adapted servers” with 2GHz dual-core CPU, 4GB of RAM, and 10 GB of available storage space such as the Raspberry Pi in a home or experimental setups.

To evaluate the performance of In.IoT in comparison with other existing solutions, a previous study that considers the performance of 5 (five) open-source middleware solutions [16] was used as a starting point. This study included Konker, Linksmart, Orion, and Sitewhere. The criteria considered by the study are error percentage, packet size, and response times. In each criterion, three scenarios were considered where data was published with 1, 15, and 100 parameters.

Apache JMeter was used to determine the packet size, response times and also generate the packets. The latest non-beta releases of Sitewhere (v2.1.0), Orion (v2.4.0, and NGSI API v2), Linksmart (v0.6.1), and Konker (v2.5.7) were used in this study. The experiments were performed in a wired LAN with 1 Gbps, ensuring that requests reach the server and any failure or delay is not due to the physical communication medium. The host Server uses a Windows 10 Intel Xeon E5-1620 v3, with 3.5 GHZ, with 4 physical and 8 logical cores. The Guest OS (where the solutions were installed) had 4 cores and 8GB of RAM. All the solutions were deployed on Ubuntu 18, with the exception of Orion, which was deployed in Centos 7.

The analysis of In.IoT error percentage will not be presented because its performance was similar to Orion, just below 0.5% across all experiments. Linksmart and Konker presented an error percentage below 1% for 1 parameter above 60% when 15 and 100 parameters when analyzed

Similar to [16], in each criterion (except for the packet size), 100 repetitions were made for each experiment.

6.4.1 Packet size to publish data

The packet size is always relevant for IoT devices because in theory, smaller messages imply reduced transmission times. A basic communication consensus is confirming a message's reception, which is known as ACK and NACK. For this reason, the reply message size is also included in the experiments. Figure 29 presents the packet size analysis for a single request. It is observed that although In.IoT and Sitewhere use JWT tokens as the authorization mechanism. The overall sum of the sent packet size by an IoT device and the received ACK message (that is sent as confirmation by the middleware) is similar across all the solutions. The exception to this rule is with 100 parameters, where devices that use Linksmart and especially Orion start sending more data.

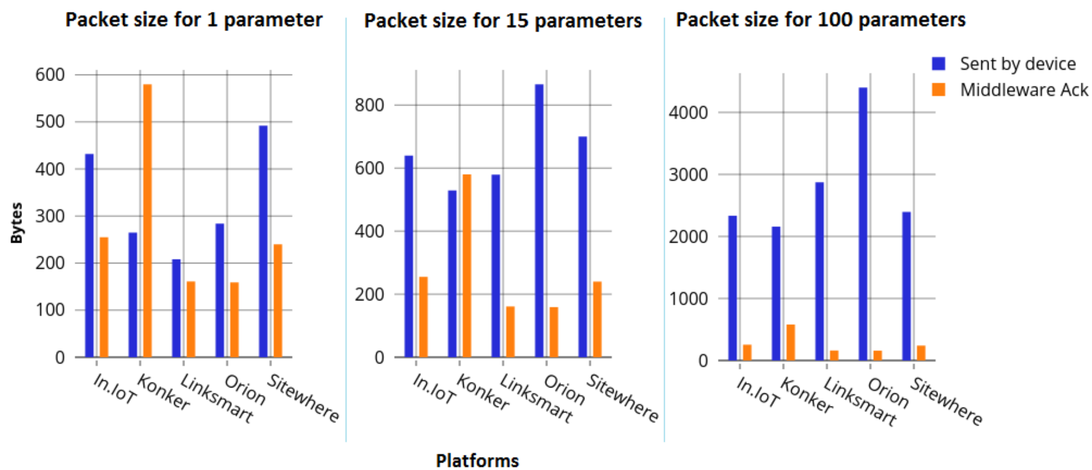

Figure 29 – *Packet size analysis of a single successful request where 1, 15, and 100 parameters were sent considering In.IoT, Linksmart, Konker, Orion, and Sitewhere middleware.*

Devices that send messages to Orion and Linksmart send more data because they have reserved words that can trigger events on the middleware. Reserved words also force the Middleware to check their presence, slowing the data processing. Both these issues can be solved by following a minimalistic approach, where only data that is intended to be stored is transmitted on the message body. Regarding the reserved words,

developers can increase performance by setting up specific routes used by the devices to trigger events, instead of processing all the messages the same way, expecting that some of them will trigger events.

6.4.2 Response times

In the context of this thesis, response times refers to the total time that the software needs to receive, process, and confirm a successful operation. Since the experiments in this chapter were performed in a controlled LAN environment, the RTT (round-trip time) was considered as the response time. In this criterium, the comparison chapter considers 5000 and 10000 concurrent users. Linksmart was excluded from the comparisons that considered a number of parameters higher than 1 because it presented an error rate higher than 15% in these scenarios. Konker was excluded from the comparison study with 10000 concurrent users because it presented an error rate greater than 15%. Sitewhere was not included in the comparison with both 5000 and 10000 concurrent users because, in Sitewhere 2.0, the minimum requirements to run the solution are 16GB of memory, which should contribute to an unfair comparison with the others. Response time is a crucial metric in latency-critical scenarios such as healthcare [254][255].

In [16], the median was used as a statistical measurement instead of the standard deviation because the experiments were performed in a real-life scenario. Then, the median is a more accurate depiction of the system behavior. Figure 30 presents the response times for 5000 concurrent users while Figure 31 presents the response times for 10000 concurrent users. In.IoT presents lower average response times than the other solutions for every experiment.

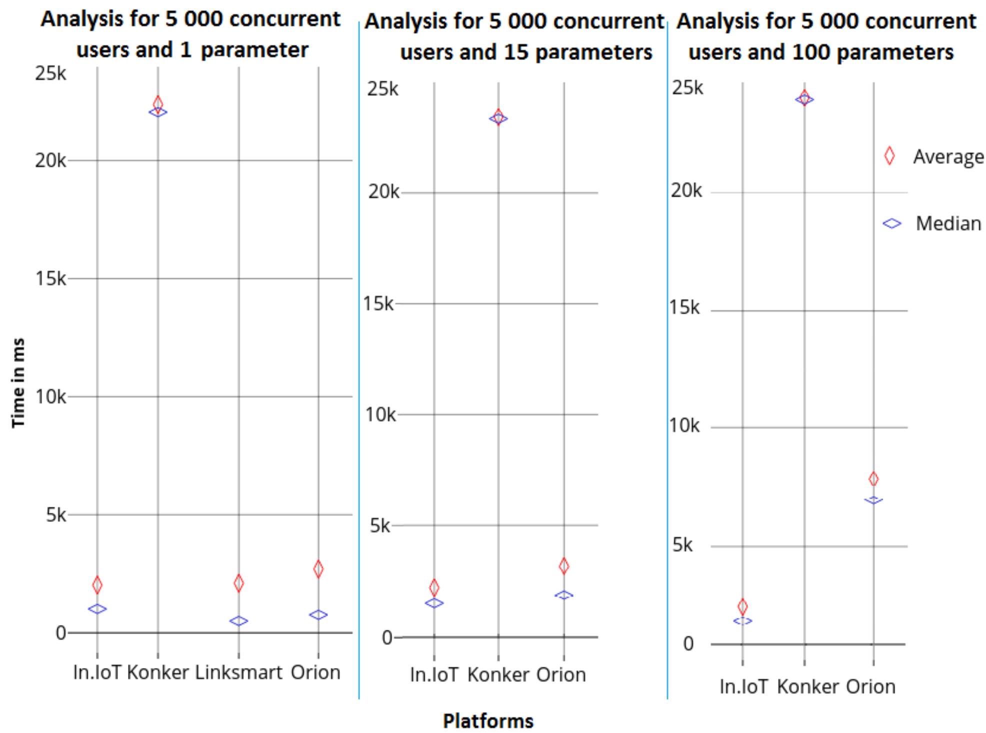

Figure 30 – *Response time analysis for 5 000 concurrent users where 1, 15, and 100 parameters were sent considering In.IoT, Linksmart, Konker, and Orion middleware*

With 5000 concurrent users, In.IoT presents a lower average response time in comparison with its competitors. For the median, it also displays lower values, with the exception of 1 parameter, where it ranks third behind Linksmart and Orion.

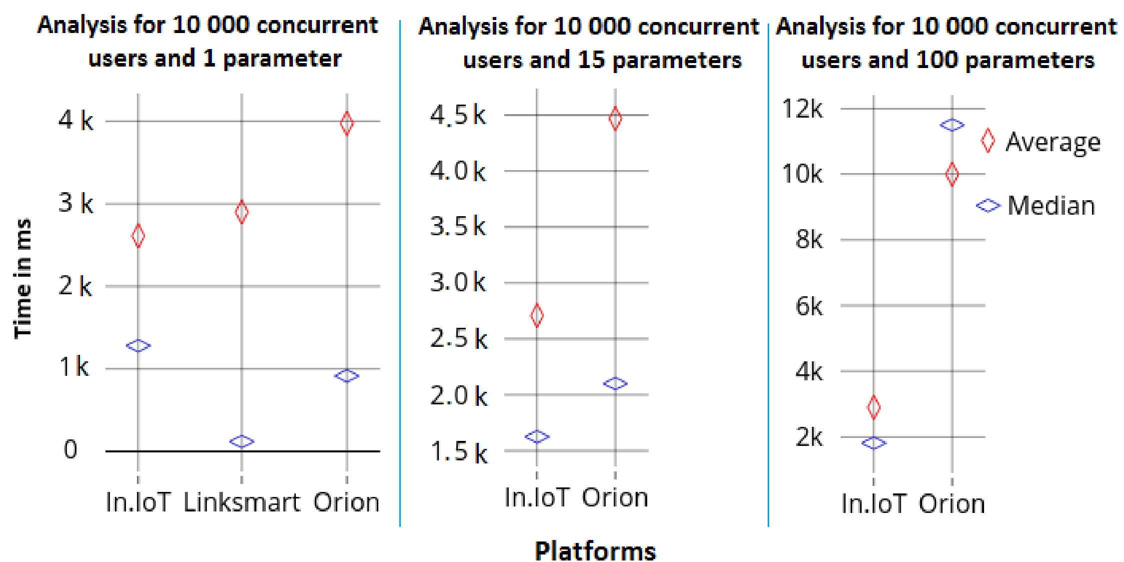

Figure 31 – *Response time analysis for 10 000 concurrent users where 1, 15, and 100 parameters were sent considering In.IoT, Linksmart, and Orion middleware.*

With 10000 concurrent users, In.IoT presents a lower average response time than its competitors. As for the median, it also displays lower values, with the exception of 1 (one) parameter, where it ranks third behind Linksmart and Orion.

6.4.3 Real-life deployment and usage

In.IoT is a generic middleware solution and is currently used in all types of IoT solutions that include smart windows, mosquito monitoring, and energy monitoring. Most of the mentioned solutions possess very specific data visualization requirements that cannot be attended by In.IoT. This issue is surpassed through other applications that connect to In.IoT and use its resources, namely its API. All the In.IoT features were successfully experimented and are continuously improved with the feedback of its users. Therefore, the proposed solution is qualified and ready to be used in any IoT environment.

In.IoT is a flexible solution that is used in various scenarios such as smart energy [256], gas monitoring [257], smart parking [258], smart homes [259], smart waste [260], and many more. The solution is patented under RPC BR 51 2018 051862-1 [261], and can be downloaded from the Bitbucket repositories [262]. Figure 32 shows an image of the In.IoT user interface that was used for a Smart energy Meter.

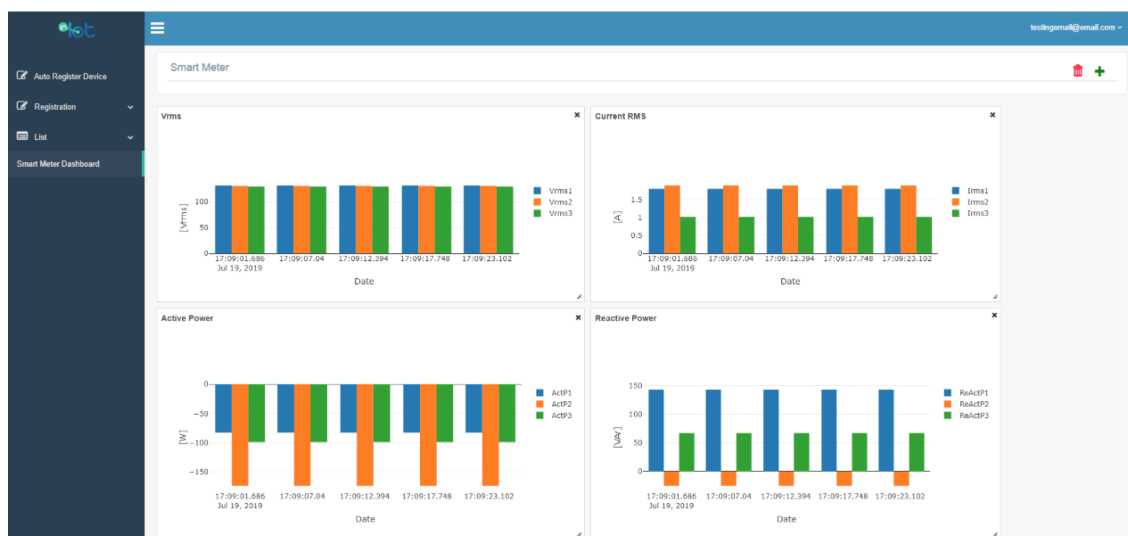

Figure 32 – *In.IoT User Interface that was used for a Smart energy Meter.*

6.5 Summary

This chapter proposed In.IoT, a new middleware platform for IoT that presents a new approach for middleware solutions and IoT devices to connect and share data. In.IoT uses a microservices architecture and its first deployment was written in Java, a proven programming language that runs on any modern operating system. The solution's efficiency was assessed and demonstrated in comparison with other existing open-source alternatives, which considered the response times, the percentage of error requests, and packet size. Furthermore, the chapter recommends security features for MQTT brokers that were implemented by In.IoT and prevent unauthorized devices from intercepting communications. The presented solution also supports firmware updates and proposes a new method for devices to securely and quickly share their abilities, while registering themselves in the platform that demands minimum effort from a human user.

The goal of In.IoT is to improve the performance and security of IoT middleware solutions. Therefore, its main features can be easily replicated by new and existing solutions that follow In.IoT architectural recommendations and requirements. The proposed solution is qualified and ready to be used in any IoT environment and currently supports MQTT, CoAP, and HTTP as application-layer protocols. A key factor for In.IoT performance is its effort to reduce the number of database accesses and various data comparisons (detailed in 6.4.4). From the authors' experience, when the number of concurrent devices is below 5000, most solutions are identical regarding performance, and if that number is never expected to increase, the functionalities offered by a middleware should be more impactful when choosing the middleware solution.

7 OLP – A RESTful Open Low-Code Platform

The ubiquitous presence and growing popularization of applications driven by the ease of access to computers and smartphones also increases software development demands for mobile and desktop applications. Corporations are looking for ways to make software development faster, more accessible, and affordable to meet these demands. When a new application (for desktop or mobile) revolutionizes, creates, or disrupts a particular market, it is not uncommon for similar applications to be introduced as competitors. The issue is that these competitors are generally introduced years after the market disruption, and a monopoly is almost established. One example is the ride-sharing giant Uber, founded in 2009, and Lyft, its biggest competitor, was founded in 2012.

The delay in introducing competitors, especially in previously inexistent markets such as ride-sharing, is mainly due to recognizing the potential market, allocating funds for the project, and development time. Big corporations generally dedicate entire teams for monitoring these market changes. They can also raise capital with little effort to acquire a disruptive company or build a similar competitor. However, these big corporations prefer to acquire these disruptive companies and only create a competitor if the acquisition does not succeed. They prefer purchasing disruptive companies because building a new application from scratch demands time and great efforts.

Building an application from scratch is challenging because an intuitive and responsive graphical user interface (GUI) is necessary for widespread adoption. This is the reason most ride-sharing apps have a similar interface. Another difficulty is that algorithms should enforce business rules and ensure user satisfaction. For example, when a ride's waiting time exceeds a certain amount of time, users get impatient and can move to a competitor ride-sharing App. Building software from scratch is also tricky because a software development team must be assembled meaning that programming languages must also be established for the team to use. Moreover, after a software team

is hired, estimating the development time is also one of the most significant software engineering issues.

A solution to reduce the software development time is the usage of low-code and no-code software development platforms [263]. These platforms do not require extensive coding knowledge and can even be used by those with no coding background to generate basic applications, which can be crucial for companies to adapt, especially in future Internet applications. Although no-code and low-code are sometimes used as synonyms, they are different concepts and can even target different user profiles. Despite low-code and no-code platforms rising in popularity, to the best of the authors' knowledge, the literature discussing how to build them is non-existent, and the advantages and disadvantages are not well-documented.

7.1 No-Code, Low-Code, and Traditional Approaches

In the early days of programming languages, it was common to program in Assembly languages similar to the target machine code instructions designed for specific computer architectures. These assembly languages are often referred to as low-level and were a nuisance because a certain machine program would only run in a machine of the same model. These difficulties led to high-level programming languages, such as FORTRAN and, later, COBOL. High-level programming languages abstract from the machine's hardware details by using a syntax closer to the human language. Initially, high-level programming was perceived with skepticism because of its bugs. However, as time passed, the programming paradigm changed, increasing software developers' productivity and made coding easier.

The trend to simplify software development continues up to now in low-code and no-code software development, which are increasing in popularity since 2014 [264]. Although their essence is the same (reduce coding and simplify software development), the two terms are not synonyms, and the difference between them is minimal and, sometimes, confusing [265]. Low-code and no-code platforms rely on visual application development without the complexity associated with traditional software development [266], improving the software development experience [267]. The main difference between low-code and no-code is that while low-code reduces code writing, no-code completely eliminates it.

The no/low-code idea derives from the fourth-generation language (4GL) concept, aiming to provide a higher level of abstraction compared to previous generations of programming languages [263]. They are often categorized into different types of domain-specific languages (DSLs), such as data management languages, database management, Web development, and many more [268]. MDSD (Model-Driven Software Development) is a concept similar to no/low-code in the sense that MDSD refers to automatic source code generation based on models and both the model and the generated source code can be debugged [269]. The main difference between MDSD and Low-code is that no/low-code platform acts as a black box where a developer has no knowledge about how the source code was generated, the used frameworks and, in some cases, even the programming languages. Low-code is also different from Integrated Development Environment (IDEs) because IDEs such as Eclipse and VSCode simplify source code writing. In no/low-code, developers mostly drag objects and hardly comes into contact with the generated source code, unless developers intend to move away from the platform but, even then, platform vendors do not always provide the generated source code. Then, in most cases, developers can only modify applications through the platform, which acts like its own programming language.

The most significant advantage of no-code and low-code approaches come from the fact that it is straightforward to build complex software, allowing organizations to quickly adapt to the ever-changing market. Also, the low and no-code learning curve is easier when compared to regular programming languages. These characteristics enable developers to quickly prototype software, translating into faster user feedback and a better customer experience. Thus, the Rapid Application Development (RAD) methodology is well-suited for this type of development, although any methodology can also be used. According to Gartner, the low-code market leaders are OutSystems and Mendix, Microsoft Power Apps, and Salesforce [270], and they are very similar regarding functionalities. In terms of main differences between them comes from the fact that some have built-in integration with external systems, such as Customer Relationship Management (CRM), social media, payment gateways, among others. According to Gartner, low-code platforms will represent approximately 65% of all development activities [271].

Like most dilemmas regarding computer science, determining which solution to use depends on the scenario and the developer's strengths. Overall, no-code is considered simpler than low-code, but this simplicity comes with the cost of less freedom for personalization in the graphical user interface and advanced business rules. Personalization limitations can be especially troublesome when integrated with external systems or using lesser-known protocols that are not supported by the platform. Therefore, it is harder to develop enterprise-grade software using no-code platforms. Furthermore, due to its simplicity, no/low-code can be used even by individuals with no programming background [272]. Although, those with a programming background are more likely to unlock its full potential.

In no/low-code development, the application development complexity is hidden by the platform [273], which is simultaneously the biggest benefit and also an issue. Platforms hide the complexity by providing a database, front-end, backend, and generate source code, which means developers can do little to nothing to maintain the developed applications if a platform is discontinued. This happens because the "source code" available for the developer consists of visual representations that are later transpiled by the platform into a source code. The lack of conventional source code can be troublesome for the developer because the platform generates and optimizes all the source code based on the visual representations. Then, it supposes the platform wrongly optimized a code block. In that case, a developer might not notice until the software has various simultaneous users. When the developers notice an inefficiency, they can only wait for an update from the platform vendor. Although advertised as reducing development costs, these platforms can be costly since their business model is generally based on monthly or yearly subscriptions.

Entirely relying on a third party without a source code also reduces the developer's ability to abandon a platform since it could mean rewriting the entire software from scratch. Therefore, if developers are not satisfied with a platform because of the existing bugs, lack of new features, or other convenient reasons, abandoning it is hardly an option. This characteristic can be especially troublesome for entrepreneurs with little coding experience that decide to jumpstart an innovative idea through no/low-code platforms. Even when an idea is a success, the platform licensing or hosting cost might not be viable, which means that it is unlikely that the entrepreneur will migrate the source code or even recover the database data. The current pricing negates one of

no/low-code biggest benefits, the ability for any person within an organization to build software since only big corporations and software houses can afford it.

Allowing any individual within an organization to build software can also present various security risks. Software developers generally have nearly unlimited access to the database, meaning that every person can potentially access confidential data (intentionally or not). This issue can be nullified through roles that limit the developer access to specific projects or even databases. Another issue is that software always has vulnerabilities, and most of them are caused by the software developer, especially those with less experience and knowledge. For example, an application developed by an inexperienced developer can gain internal popularity (i.e., it is widely adopted within the organization of the user that created it), and its developer might not even be encrypting passwords. Since most users tend to repeat passwords, this simple oversight could have severe and dangerous consequences.

In the early days of low-code, the personalization level compared to traditional software development was very lackluster. Nowadays, some platforms such as OutSystems allow developers a high degree of personalization [274]. Overall, the pricing of no/low-code platforms is a significant barrier to its popularization. Therefore, each corporation should evaluate the benefits and drawbacks of using this type of solution for software development. It is hard to imagine that no-code can reach a degree of personalization comparable to traditional programming, but this was also an issue for low-code in the past. The next evolution of the no/low-code concept will likely be software developed by artificial intelligence (AI) solutions where users verbally explain what is desired.

As previously mentioned, no/low-code platforms are very similar, and their characteristics are summarized in Table 4.

<table>
<tr><th>No-code Characteristics</th><th>Low-code Characteristics</th></tr>
<tr><td>Removes code writing</td><td>Reduces code-writing</td></tr>
<tr><td>Low freedom for personalization</td><td>High freedom for personalization</td></tr>
<tr><td colspan="2">Easier learning curve when compared to traditional programming, allowing anyone to build software</td></tr>
<tr><td colspan="2">Visual application development</td></tr>
<tr><td colspan="2">Code optimization</td></tr>
<tr><td colspan="2">Hosting and/or licensing costs</td></tr>
<tr><td colspan="2">Generally hides the true source code from the user</td></tr>
<tr><td colspan="2">Less knowledgeable or inexperienced developers are likely to create vulnerabilities due to the ease of creating applications</td></tr>
</table>

Table 4 – *No/Low-Code main characteristics.*

Since this chapter focuses on low-code, only concepts associated with low-code will be addressed, although most of the low-code concepts can also be applied to no-code.

Any software project is built based on requirements, which can be functional or non-functional. Non-functional requirements focus on the features that applications should provide to ensure Quality of Service (QoS) and they are presented in subsection 7.1.1. Functional requirements describe which application features should be deployed, and are presented in Subsection 7.1.2. The proposed architecture is showcased in Subsection 7.1.3

7.1.1 Non-Functional Requirements

The proposed solution addresses the following non-functional requirements:

1) **Ease of use**: Since the primary goal of this type of software is to simplify software development, platforms should be intuitive to handle. In practice, intuitive means that users should be able to develop applications with minimal training and effort. This low complexity is achieved by replacing traditional source code with visual representations that allow users to manipulate system elements graphically instead of textually. This crucial change is associated with a concept known as visual programming, allow developers to focus on software functionalities, and it is also good for educational purposes [275].

2) **Flexibility**: Since this software enables users to build other software easily, the platform must be intuitive. Nevertheless, it should support advanced use-cases and this is only possible if the platform provides a degree of flexibility to users.

3) **Extensibility**: The platform should provide tools or documentation that allow users to refine or extend the provided functionalities through plugins or other contributions. This feature is essential because no matter how flexible the solution is, it is unlikely to attend all use cases.

4) **Interoperability**: This software should be able to integrate with external systems [276]. Nowadays, integration with other solutions is generally accomplished through REST (Representational State Transfer) APIs (Application Programming Interface).

5) **Security**: It is common for programmers having almost unlimited access inside an organization and with access to privileged information (in smaller organizations, this is even more common). Therefore, low-code platforms should

provide security mechanisms that can limit access to the source code and allow accountability. In addition, similar security mechanisms should be easily applied to the developed software, allowing profile access and methods to encrypt fields within a database securely.

6) **Privacy**: The concept of privacy is tightly linked to security because it focuses on methods and purposes of storing, analyzing, and sharing data by a service provider. For example, a recent privacy controversy was sparked by WhatsApp, which started sharing user data with its parent company Facebook. Users were fearful that their private conversations would be shared and used for a targeted advertisement within Facebook, Instagram, or other third parties. Facebook later reiterated that not even WhatsApp could access private conversations due to end-to-end encryption, which meant that such a scenario would not be possible. Knowing which data is shared and how data is stored could be especially useful in a data breach because various data could be compromised directly or indirectly. General Data Protection Regulation (GDPR) from Europe [277] and California Consumer Privacy Act (CCPA) from California, USA [278] are examples of privacy protection legislation.

7) **Scalability**: The platform should provide tools or be compatible with tools that allow software to scale vertically and horizontally, adapting to the ever-increasing demands. Vertical scaling consists in running a solution in hardware with more resources (memory, HDD, CPU cores, and processing power) [279]. Horizontal scaling is characterized by distributing the workload through various hardware [280] instead of having a single server with powerful capabilities, the workload is distributed among several servers.

8) **Maintainability**: Low-code platforms are constantly updated by their developers, which means that a platform should be constructed in a way that is easy to maintain. Maintainability is an attribute that describes the simplicity of modifying, fixing bugs, or improving the performance in a software solution.

9) **Backwards compatibility**: Low-code platforms continuously iterate and improve themselves, especially regarding usability while trying to not being so strict that too much flexibility is lost. When visual code becomes incompatible with its previous implementation, the platform should automatically upgrade it to end-users. When done correctly, even visual code from version 2 of a platform that is incompatible

with a version 6 can be upgraded to version 3, which can be upgraded to version 4, and so on.

7.1.2 Functional Requirements

The functional requirements are the following:

1) **Data management and event management**: A software should allow "create, read, update, and delete" (CRUD) operations performed to a database or an external API. When manipulating data directly from a database, users should be able to perform advanced database query operations. These operations will generally require users to write SQL (Structured Query Language) or even NoSQL statements. Furthermore, modern systems usually execute or respond to various events automatically, according to the received data, and the platform should also provide such as a feature.

2) **Code transpiling**: Since the source code consists of visual representations instead of the traditional text, visual representations must be translated into a textual source code. Then, it can be written in any programming language chosen by the platform developers and should be compiled or interpreted depending on the chosen programming language.

3) **Correctness**: The generated textual source code should not contain errors or bugs, since most platforms do not offer users the source code. Thus, users will not be able to edit code that is generated by the platform. This is valid for all the coding aspects, including a REST API or a script to generate a database.

4) **Code optimization**: The generated code should be optimized because, otherwise, the software that it produces might be difficult to scale. Also, the generated source code should be easily readable by the platform developers because it is likely they will need to review it in the future.

5) **Code verification**: All the generated code should be verifiable through automated tests, this is valid for the platform itself (when it is being built or modified by the developers) and also for the end-users. The developers need to use or build code verification tools because modifications to platform code can alter internal functionalities and negatively impact functionalities on the end-user application. The platform should also provide similar tools for the end-users so they can verify their functionality, detecting bugs of their own-doing and also bugs because of updates in the

platform. Furthermore, automated testing is a good practice in software development that saves time in the long run and ensures product quality [281].

6) **Code compiling/interpreting**: After the transpiled source code is verified as being correct and optimized, it should be compiled or interpreted depending on the programming language. This is a necessary step in most programming languages because, otherwise, running the application will not be possible.

7) **Application deployment**: Deployment is a process that makes a software available to its end-users and can vary depending on the targeted end-user. In a mobile application, a deployment could mean publishing the software on an App store or running the application on a smartphone. In Web applications, it could mean running on a local computer or on a cloud server.

8) **Produce adequate messages**: A big part of programming is the textual output produced by compilers, which are generally categorized by i) information, ii) warnings, and iii) errors. These textual outputs are essential because they give the programmer important feedback regarding their code. Generally, errors mean that code will not compile, and warnings imply that some aspects could be optimized or might not work as expected. An example of a warning or error (it depends on the compiler) could be the following: variable bar in function foo (Float bar) expects a Float but receives an Integer when called. The previous example is not a suitable warning because there is no reference to the file where the error occurred or the line it refers to. Since low-code movement goals to facilitate programming, the location where the errors or warnings occur should be easy to identify, and the user should easily interpret them.

9) **Debugging functionalities**: Even the best software present bugs and debuggers enable software developers to identify and remove the source of such bugs. Furthermore, debuggers allow software to run in a completely controlled environment. In practice, this means that each line of code can only be executed after programmer presses a button. In addition, the debugging process enables developers to identify and monitoring changes in various resources such as variables and should be avoided in production environments.

10) **Visual-code export/import**: The visual code that developers use should be easily exported (saved) enabling users to import it (load) without losing functionalities.

11) **Textual code writing**: Low-code platforms should offer ways to manually write source code because it is one of its core functionalities (minimize code-writing,

but not eliminate it). However, allowing developers to write traditional source code, the platform should also have a well-defined syntax and semantic.

12) **Versioning and collaborative development**: Most software developers verify the impacts of their code changes immediately and, with low-code, this aspect will be the same. In visual programming, it could be easy to modify or remove visual code blocks and lose functionality, which could worsen if the erased block contains textual code. Therefore, these platforms should provide version control so developers can go back to a previous version. Furthermore, version control allows multiple developers to simultaneously work on the same project without breaking each other's code.

13) **Visual data modeling tools**: Users should visually configure and assert constraints when creating entities, their fields, dependencies, and even relationship with other entities.

14) **Visual programming tools**: Since low-code programming will minimize code-writing, it should rely on visual representations that allow users to control program elements graphically. Among other aspects, these tools should offer users the ability to represent the most basic aspects of any software: i) sequences, ii) selections, and iii) loops [282]. It consists in a sequence of actions completed in a specified order; in traditional programming, it executes each line in sequential order (e.g., line 5 only executes after lines 1, 2, 3, 4). Selections formulate questions to decide which subsequent lines to execute. Loops are similar to selections because they continue to formulate questions and execute subsequent code until a specific condition is reached.

7.1.3 Architecture

Based on the presented requirements, the architecture of the low-code platform considers the following six main components: 1) visual application modeler, 2) encoder, 3) decoder, 4) source code generator, 5) compiler, and 6) deployer. These elements are presented in Figure 33 and are described as follows:

1) **Visual Application Modeler**: The front-end of the low-code platform will interact with the developers and simplify software development. The visual application modeler is an enhanced Integrated Development Environment (IDE) that implements most of the functionalities that developers interact with. These include producing code (graphical or textual), debugging, modeling data, code verification, testing, versioning,

event management, and many more. The IDE should be simple to use [283], provide a preview of the developed software, and is located at the client-side.

2) **Encoder**: The "symbols and codes" used in the visual application modeler must be imported and exported without losing functionalities. To achieve such a goal, the "code" produced by the modeler must be expressed in a self-contained representation easily interpreted by an algorithm. This entity encodes the visual representations into a flexible format such as JSON (JavaScript Object Notation) or XML (Extensible Markup Language) that can be transmitted over the Internet. The encoder is located on the client-side.

3) **Decoder**: This entity interprets the encodings of the visual representations and is located at the server-side.

4) **Source code generator**: After the data is represented in a human-readable way, it can be transpiled by the source code generator. Transpiling consists of converting the source code from one language to another. This entity should include scripts to generate and interact with a database and it is also responsible for code correctness, optimization, and verification.

5) **Compiler**: Is the entity responsible for compiling the source code and also acts as the final code correctness test.

6) **Deployer**: Is the entity responsible for deploying the software into a target platform that will interact with end-users.

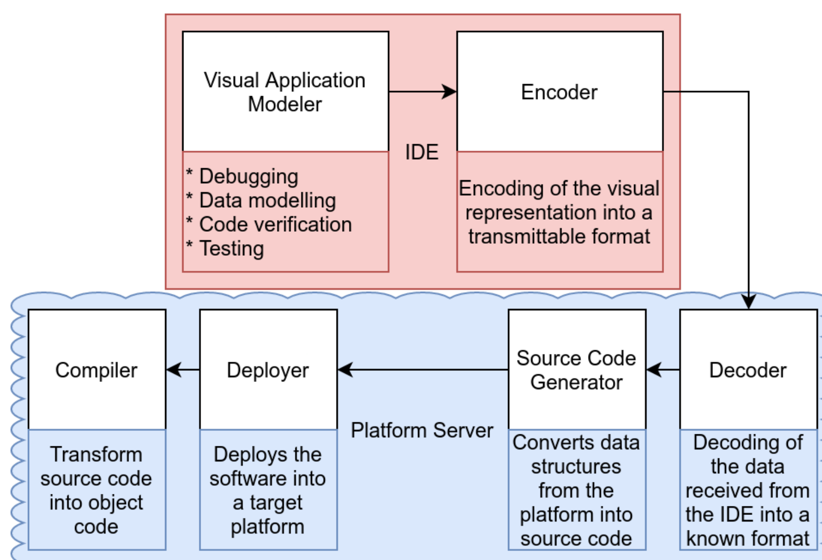

Figure 33 – *The general architecture of low-code platforms.*

7.2 OLP Development and Demonstration

Developing a low-code platform from scratch is challenging because of the little available literature and architectures are not always easy to interpret in practical scenarios. To facilitate the development of future low-code projects, the Open Low-Code Platform (OLP) was created. The scope was limited to Web applications development and data was persisted and consulted through REST APIs. To develop OLP, the authors drew inspiration from developing a new programming language, and the first task was developing a syntax and, consequentially, the operators precedence. Since graphical visualizations and written source code must have syntax, it was first represented in EBNF (Extended Backus-Naur). Formalizing a syntax is helpful because the platform developers can consult it at any time, enabling productivity and cooperation. EBNF is a meta-syntax used to describe other syntaxes. A logical or arithmetical expression representation in EBNF is presented in Code 1. The other representations used in OLP can be found on the GitHub repository along with its source code [284].

Code 1. EBNF representation of an expression.

```
Expression
  = primary
  , [ binary-op , expression ]
  ;
```

After establishing the syntax, the developers should define a semantic that will enable the platform to differentiate between coherent and non-coherent statements. An example of an incoherent statement could be dividing a String by a number. Finally, a pipeline that details how the code will be transformed from the visual representations to a written source code was established and it is presented in Figure 34. Such pipeline was based on the architecture presented in Figure 33 and considers two parts, IDE and code generator.

The IDE supports most of the functions that end-users will interact with and simplifies software development. These functions include producing code (graphical or textual), debugging, code verification, testing, versioning, event management, etc. The IDE also provides a preview of the software being developed by end-users. In OLP, the IDE is browser-based and was built using AngularJS. A browser-based IDE was chosen because web browsers allows the HTML (HyperText Markup Language) on the screen

to be inspected, OLP takes advantage of this feature and extracts it, which means that the HTML is obtained directly from the browser. In addition, the code generator will translate the visual representations into source code in another programming language. When building the code generator, control flow structures frequently found in imperative languages were used because this enabled structures with a higher level of abstraction to be defined. In OLP, the code generator was built using Haskell.

In Figure 34, the upper division of the pipeline describes the processes that occur at the IDE and the bottom half describes the processes that occur on the code generator. The left side of the pipeline describes the process of analyzing and converting a low-code program into a JavaScript file and it is called scripting pipeline. The right inside is called User Interface (UI) pipeline and describes the process of converting the visual representations built on the IDE into CSS (Cascading Style Sheets) and HTML files. The scripting and UI pipeline occur simultaneously and are described as follows:

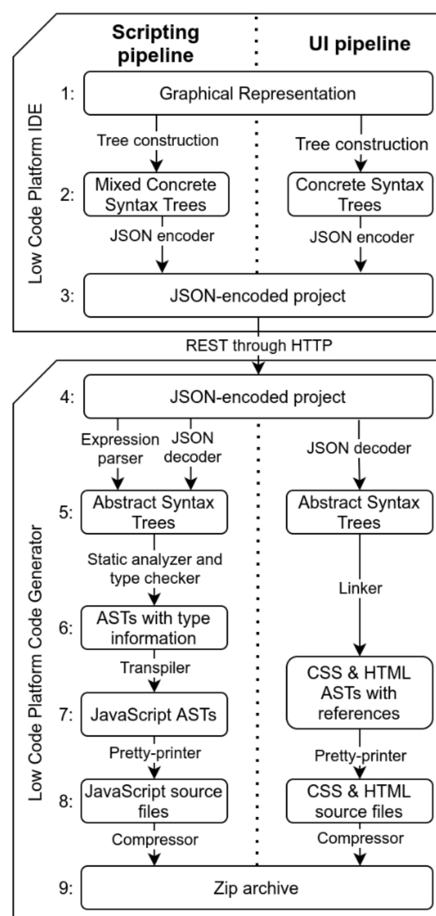

Figure 34 – *Pipeline detailing how the code is transformed from the visual representations into a fully-fledged application.*

1: The user exports the project and transforms the visual representations into a format that can be easily encoded and decoded for future usage.

2: Programming languages generally represent data structures in compilers through a tree data structure because it resembles the production rules for syntax. In the case of the platform, an Abstract Syntax Tree (AST) was chosen to make this representation. On the scripting side (left side) of the pipeline, the AST may contain strings, because even though it's a scripting language, the user is free to type any expression in text boxes. These expressions typed by the users must be parsed before being converted into a pure AST. The other side of the tree is already in the correct format as the visual representations can be converted directly into objects. In the UI (right side) of the pipeline, the tree is already assembled in the correct format because the user cannot type custom HTML.

3: The AST is converted to JSON because the AST needs to be represented in a format that humans can understand, enabling faster debugs. Then, an HTTP request is made to the compiler, where the encoded JSON is sent to the compiler.

4: The Compiler receives the JSON containing the encoded AST as well as other project data (name, the user who created it, settings, and other files).

5: On the logic side (left side of the pipeline), the JSON is decoded normally except in the parts that were typed by the user and are not verified by the IDE and need to be analyzed by the semantic analyzer. The trees are already set up on the UI side (right side of the pipeline), so they are converted directly to ASTs.

6: On the logic side (left side of the pipeline), the semantic analyzer places data type information in the tree. It also looks at some other inconsistencies and generates warnings. This step is exclusive to logic.

7: On the logic side (left side of the pipeline), the transpiler converts the AST from step 6 into a JavaScript AST. Aspects such as data types that are supported by the platform but do not explicitly exist in Javascript are converted in this step.

8: The pretty-printer transforms ASTs into a styled and formatted source code that is easier to read. The source code is formatted and styled for the sole purpose of faster debugs and it is the same on both sides of the pipeline.

9: The linker references the CSS and javascript files in the HTML and compresses them into a Zip file.

Most of the elements from the architecture presented in Figure 33 can be found on the pipeline. The Visual application modeler represents step 1, the encoder occurs on step 2, and the decoder on step 3. The source code generator was deployed in steps 4-8. The compiler from the architecture presented in Figure 33 was not included because

Javascript and HTML source code does not need to be compiled (they are interpreted by the browser). The deployer will be implemented in future releases.

7.2.1 Demonstration

OLP is a low-code platform built to better understand the practical difficulties of developing a platform from scratch. Its scope was limited to Web applications development, with data persistence and consultation through REST APIs. Unlike existing platforms, the code generated is available to the users at any time, which means that users can resume development if the platform stops being updated. Although the proposed solution was built as a proof of concept, it is qualified and ready for use. The platform is also open-source software with a permissive license, allowing any user to contribute to its development or maintain its own fork. The code generated by the platform means that any user can continue maintaining his projects, even if the project stops being supported. The solution and its source code can be downloaded on the GitHub repository [284]. Figure 35 shows the IDE for an application that pulls the current temperature and displays it on a canvas. Figure 36 shows the logic part of such an application. Figure 37 shows the main function of the source code generated by the platform.

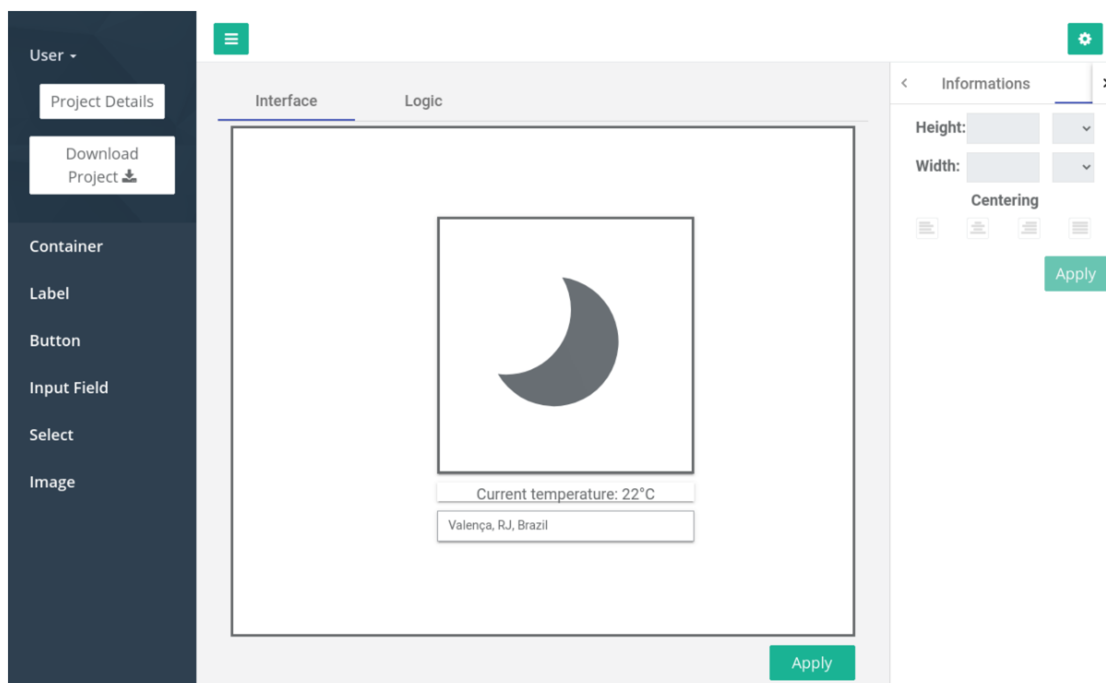

Figure 35 – *Interface creation screen for an application that pulls the temperature from a website and displays its value on a canvas.*

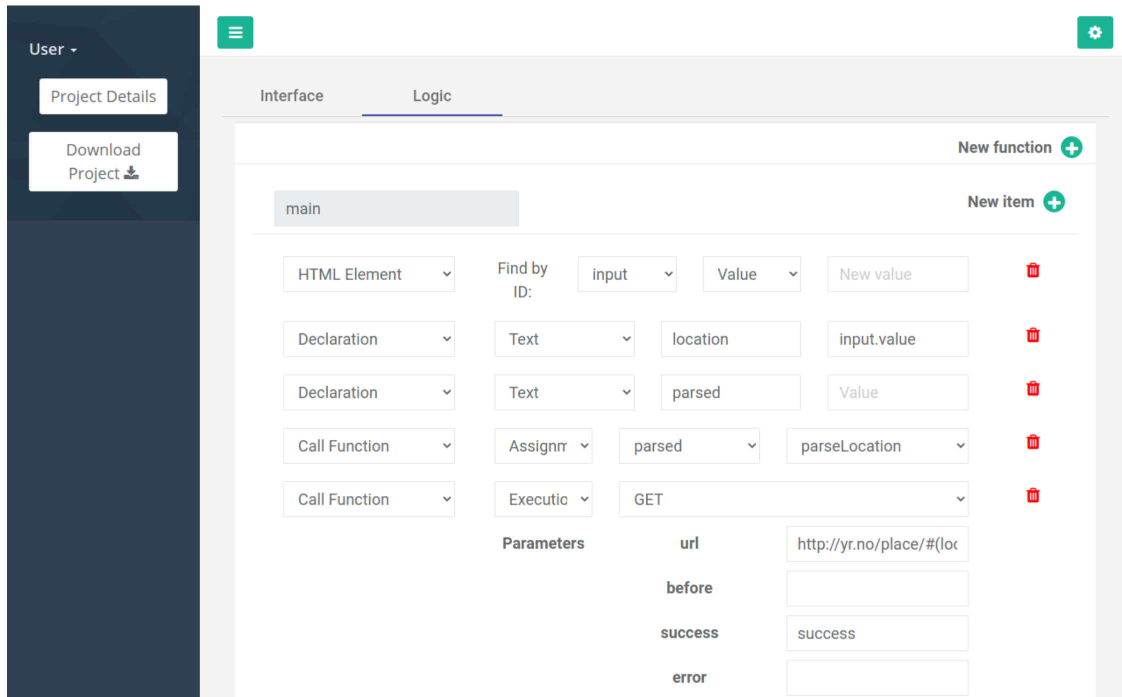

Figure 36 – *Logic screen for an application that pulls the temperature from a website and displays its value on a canvas.*

```
1 "use strict";
2
3 function main() {
4     let location = input.value;
5     let parsed;
6     parsed = parseLocation(location);
7     GET(`http://yr.no/place/${parsed}`, () => {}, success, (_0, _1, _2) => {}, () => {});
8 }
```

Figure 37 – *The main function of the source code generated by the platform from the steps presented in Figures 35 and 36.*

7.3 Summary

This chapter described a low-code concept which consists of transforming visual representations that are easy to understand into fully-fledged applications. The functional and non-functional requirements of low-code, the similarities with the no-code concept, and how both concepts are different from traditional programming are also explored. The chapter also recognized the difficulty of developing a low-code platform due to the lack of available literature and proposed the Open Low-Code Platform (OLP), a new low-code platform. OLP is a low-code platform built to understand the practical difficulties of creating a low-code platform from scratch. The

details of how the visual representations can be transformed into an application are also presented and explained through a pipeline.

Low-code platforms allow full applications to be developed extremely fast, which is crucial in today's world and also in the future, with time to market becoming increasingly important.

8 Detecting Compromised IoT Devices Through XGBoost

Currently, many IoT devices are vulnerable because there is always a tradeoff between security and usability. Unfortunately, IoT manufacturers tend to maximize usability, fearing that average users will lose interest if a device cannot be used straight out of the box. This lack of optimal device security generally reflects on IoT networks and platforms, endangering the whole environment and jeopardizing the IoT concept's widespread adoption [285]. Security is a crucial element of any system and devices connected to the Internet demand additional precautions because threats can arrive anytime from any part of the globe.

Most of the vulnerabilities are relatively easy to prevent, especially when the best practices are followed. However, even when most device vulnerabilities are mitigated, the replication attack is hard to detect and prevent. In the replication attack, intruders obtain device credentials (legitimately or not), potentially disrupting an IoT network or leaking data. Then, detecting the occurrence of such an attack could vastly improve the security in IoT environments. Since IoT middleware is responsible for a significant part of the intelligence in IoT environments and it is located on a powerful server [15], they could use additional resources to detect such an attack. In this sense, the chapter aims to detect such attacks through a machine learning technique known as the XGBoost.

Machine learning (ML) is an excellent tool when a problem requires discovering hidden patterns from a large dataset. A common issue in ML happens when a model becomes so familiar with the training data that it can no longer adapt to other data, then it is said the model is overfitted. When a model cannot identify the patterns, even in the training set, which translates into even worse generalization, then the model underfitted. The chapter goals to detect the replication attack where an attacker obtains device credentials and uses them to disturb the IoT environment. In this sense, XGBoost was

the chosen ML technique because it shows excellent results in various areas and is seen as an evolution of the also popular random forests and decision trees, which also produce accurate results with labeled data.

8.1 Security threats in IoT environments and similar studies

This subsection provides a review of the most relevant and common attack threats in IoT environments that directly affect devices and software platforms, as well as the recommended methods to mitigate them.

8.1.1 Security threats for IoT platforms and networks

Networks are critical in IoT systems because they provide connections for devices. Combined with IoT platforms, networks allow devices to interact with applications and deliver storage and advanced computation capabilities. It is a consensus that IoT networks should only disclose data to authorized entities [286] because this type of data leakage could have unforeseen consequences. In this sense, the most common security threats for IoT networks and platforms identified in [287] are the following: *a*) sniffing attacks, *b*) Denial of Service (DoS) attacks, *c*) spoofing attacks, *d*) routing attack, *e*) IoT cloud service manipulation, and *f*) privilege escalation. Next, each security threat is briefly described as follows.

a) **Sniffing attack**: The attacker monitors and captures packets that are sent or received by a given network or host to obtain credentials or other sensitive data. The tool that is used to capture network data is called a packet Sniffer. Packet sniffers are not always malicious tools because they can enable network administrators to diagnose network issues. Two popular packet Sniffers are *Observer* and *Wireshark* [288].

b) **Denial of Service (DoS) and Distributed DoS attacks**: An attacker floods a network or a host with huge amounts of data, causing entire systems to become unavailable. From the perspective of a server, a DoS is relatively easy to counteract since it involves a single machine and blocking the IP address of the attacker can nullify the threat. However, mitigating a Distributed DoS (DDoS) in which multiple devices perform DoS attacks on the same target [289] is a much more complex task.

c) **Spoofing attack**: The attacker impersonates a network entity to receive compromising data such as credentials from one of the legitimate entities. Attackers can

spoof an IP address in scenarios where the IP is used for access control and for gaining various network privileges. In RFID solutions, attackers can clone data from a legitimate RFID tag and impersonate it. The Man-in-the-middle (MitM) attack, where the attacker impersonates both the sender and receiver, is the more advanced Spoofing attack.

d) **Routing attack**: The attacker changes how packet routing is performed on an IoT network, generating routing loops, fraudulent error messages, or even obtaining sensitive information. A simple routing attack can consist of a node advertising itself as the shortest path to a common network destination [290], then *i*) dropping all packets, or *ii*) forward data to a malicious server. In *i*), the attacker effectively stops the network. In *ii*), the attacker can obtain sensitive data such as device credentials.

e) **IoT cloud service manipulation**: The attacker gains control or access to one or multiple cloud services. When this attack is successful, the intruder can obtain data directly from the database or even disrupt an IoT environment by generating false alerts. An example of such is an exploit in Docker containers that could only be executed with administrator privileges for a reasonable amount of time, which meant that a malicious process inside the container could inherit its privileges. However, recent Docker versions allow containers to be deployed without administrative privileges to reduce the impact of such an attack.

f) **Privilege escalation**: The attacker gains access to a low-level account (legitimately or not) to access other protected services. The premise of the attack can be understood by the following example: a company that uses keycards as an access control mechanism only allows visitors to access the common rooms. However, a visitor notices that any subsequent room is accessible with the visitor keycard once inside a restricted area. Then, he/she proceeds to exploit this vulnerability and access other restricted areas.

The threats to IoT platforms are generally mitigated through *i*) authentication and authorization, *ii*) traffic filtering and firewalls, and *iii*) encryption protocols.

i) **Authentication and authorization**: Most Web applications require users to provide their credentials, generally through a username and password combination. This action is called authentication and allows the server to certify that the user is registered. After the authentication process, the server verifies which actions can be performed and it is called authorization. Authentication can also be simplified as an answer to "who are you?" and authorization as "what can you do?". A well-configured authorization can mitigate privilege escalation as well as IoT cloud service manipulation attacks.

ii) **Traffic filtering and firewalls**: A firewall is a network entity (software) that analyzes the network traffic and decides to block or allow traffic based on previously configured security rules [291]. A well-configured firewall can mitigate various attacks, especially from known threat sources. However, firewall rules need continuous increments, especially in an IoT context.

iii) **Encryption protocols**: These protocols ensure data confidentiality, so users that "eavesdrop" unauthorized communications cannot decode data. Most encryption protocols make use of symmetric and asymmetric key management. One of the most popular protocols on the Internet is HTTPS (HyperText Transfer Protocol Secure) because of the added security. HTTPS's security relies on TLS (Transport Layer Security), a cryptography protocol that uses a private and public key pair to encrypt communications. In IoT communications, cryptography protocols could be used to encrypt MQTT (Message Queuing Telemetry Transport) and CoAP (Constrained Application Protocol) communications. Without encryption, device data is transmitted in plain text, which means that any attacker that intercepts it can read the message.

8.1.2 Security Threats for IoT Devices

A device is likely the most fragile component of IoT domains. They are mainly limited regarding battery life and computational capabilities because security contrasts with performance. A compromised node can compromise the entire network by generating erroneous measurements or even carrying out malicious network attacks. In this sense, the most common attacks related to the device layer identified in [287] are the following: *a*) Hardware Trojan attacks, *b*) Replication attacks, *c*) tampering attacks and malicious code injection, and *d*) battery-draining attacks.

a) **Hardware Trojan attack**: This is a process in which a malicious action is purposely inserted in the circuitry during the assembly or construction phase [292]. This action stays hidden and is unnoticed until it is triggered by the manufacturer or other third party aware of the action. In other words, it is a backdoor that is added during the manufacturing process. When a device is compromised during the manufacturing process, the issue might not be solved if the vulnerability is placed directly on the hardware itself. There is no real way to prevent or mitigate such an attack.

b) **Replication attack**: This is a process in which attackers insert a device with the same credentials as a target device, generating false data or obtaining other security grants like cryptographic shared keys. This type of attack requires that attackers possess

the credentials (generally username and password) of a legitimate device and can be hard to detect. This attack can also threaten IoT platforms because once the device credentials are obtained, the attacker can gain a degree of access to IoT platforms. This attack cannot be prevented, but it can be mitigated with a proper detection mechanism.

c) **Tampering attack and malicious code injection**: This attack generally occurs when an attacker obtains physical access to a device, inserts malicious code, or retrieves the device logs [293]. This type of attack can also be remotely performed by exploiting the default or commonly used username/password combinations and uploading a changed version of Firmware. *Mirai* is an example of malware frequently used for this type of attack that can hijack IoT devices remotely and create botnets to perform distributed denial of service (DDoS) attacks [294].

d) **Battery draining attacks**: It consists in reducing a device's battery life by continually sending data to it and reducing its "sleep time" or forcing constant reply messages [295].

Most of the threats to IoT devices can be mitigated by limiting physical access to the IoT devices. Other recommendations that can mitigate IoT device threats are the following: *i*) individual device credentials, *ii*) mechanisms to modify device credentials, and *iii*) source code protection.

i) Individual device credentials: IoT devices should have an individual username and password combination. Otherwise, an attacker can disrupt an entire IoT network in an unimaginable manner by obtaining a single devices' credentials. *ii*) Mechanisms to modify device credentials: The ability to modify device credentials refers mainly to the password. This is important because there is always a possibility that an attacker can obtain a password. *iii*) Source code protection: A device source code determines its operation rules and various constraints. Thus, it should be protected from external access through source-code protection mechanisms that guarantee that source-code cannot be retrieved and alert users (administrators) when new source-code versions are uploaded to the device.

8.1.3 Similar studies

The only threats without a clear counter are the hardware trojan attack (which cannot be countered) and the replication attack, which can only be mitigated with a proper intrusion detection mechanism. In this sense, the two main approaches to detect

intrusion consist of signature detection and anomaly detection. Signature detection techniques store the unique characteristics of known attacks in databases to compare with the incoming traffic. Anomaly detection usually uses machine learning to detect hidden patterns on the incoming data and differentiate legitimate traffic from malware traffic.

Signature-based detection is excellent for identifying known threats but can be ineffective against new attacks whose signatures are not in the database. Signature-based detection requires constant maintenance because the new signatures must be continuously inserted into the database. Regarding anomaly detection techniques, since they mostly use machine learning, a model must be trained based on a dataset to detect the anomalies. A dataset used in recent studies is IoT-23 [296], because it is available to the public and contains labeled benign and malicious IoT network traffic captured from 2018 to 2019 [297]. The dataset labels several threats like Okiru, Torii, Mirai, port scanning DDos, C&C, and heartbeat. The issue with IoT-23 is that it is a skewed dataset because most of the recorded data is malicious.

In a recent study, [298], a security framework called ADEPT was created to detect suspicious activities in the network. ADEPT uses a combination of K-NN (K-Nearest Neighbor), Random forests (RF), and SVM (Support Vector Machine) algorithms. Furthermore, the study also used three datasets: UNSW-NB15 [299], NSS Mirai Dataset [300], and IoT-23. After training the model, the experiments containing data from the three datasets showed great promise, with ADEPT successfully identifying attacks with diverse characteristics.

In [301], the IoT-23 dataset was used to train two distinct models, the first model was based on Multi-Class Decision Forest and the second was based on a Multi-Class Neural Network. In both models, the percentage of true negatives was 100%. Regarding the percentage of true positives, the multi-class decision forest was 99.8% and the multi-class neural network was 99.7%. In [302], an ensemble method using a DNN (Deep Neural Network), LSTM (Short-Term Memory) and logistic regression as a meta-classifier was used for intrusion detection. The work demonstrated the efficiency of such an approach by using data from the IoT-23, LITNET-2020 [303], and NetML-2020 datasets [304].

In [305], a lightweight bot detection mechanism called BotFP was created, and three variants were evaluated: BotFP-Clus, BotFP-MLP, and BotFP-SVM. BotFP-Clus was trained with the DBSCAN (Density-Based Spatial Clustering of Applications with

Noise) algorithm. BotFP-MLP used Multi-Layer Perceptrons and BotFP-SVM used Support Vector Machines. The study uses the CTU-13 dataset [306] combined with the IoT-23 dataset to evaluate each model. The results were promising, considering BotFP successfully detected all bot occurrences on the CTU-13 dataset and few false positives.

The chapter will focus on detecting abnormal network traffic to mitigate the replication attack by finding hidden data patterns. In this sense, the chapter will use ML because it is an excellent tool to identify hidden patterns in data. An XGBoost model will be trained using data from the publicly available dataset IoT-23. XGBoost was chosen as the ML technique because it is obtaining great results on Kaggle competitions [307], which includes tasks from Kaggle and some big companies like Google, demonstrating its flexibility across various scenarios.

8.2 XGBoost

In areas like e-commerce and social media, traditional software is becoming obsolete because clients expect to view information relevant to their preferences, which is generally achieved through machine learning (ML) [308]. A machine learning approach considers machines (typically computers) making predictions and improving their accuracy based on the analyzed data [309]. Machine learning techniques are good for finding hidden patterns in data but require large amounts of training data to generate models that produce accurate results [310]. Luckily, datasets on various topics are available on the Internet and can be used for research, free of charge.

Recently, a ML technique, called XGBoost, has become increasingly popular because of its immense benefits. XGBoost uses gradient boosted decision trees, which means that understanding XGBoost, implies understanding decision trees. Decision trees are supervised models that best predict the result with only two possible outcomes (e.g., spam or not spam). Decision trees can also be applied to non-binary outcomes, but such an approach increases the complexity. Decision trees are visually represented as binary trees where a dataset is broken into smaller subsets, called nodes, as the tree grows, which represent true or false statements [311] based on variables from the dataset. The main issue with decision trees is that the order in which the variables are

analyzed (questions are formulated) can change the outcome. Thus, the first variable to be analyzed should provide more information gain than the others in the sequence [312].

A typical application for decision trees is predicting the outcome of an election based on known variables, such as education, income, and diversity. Depending on the data in each feature variable, the tree's root should be different because the variable located in the root should contain more information. Figure 38 shows a decision tree that determines the probability of a state voting for a yellow or a purple party, and education was chosen as the tree's root. When a variable cannot be represented as a Boolean (false or true), the numerical data is split into thresholds that determine if the result is more likely to go left or right (false or true in Figure 38 example).

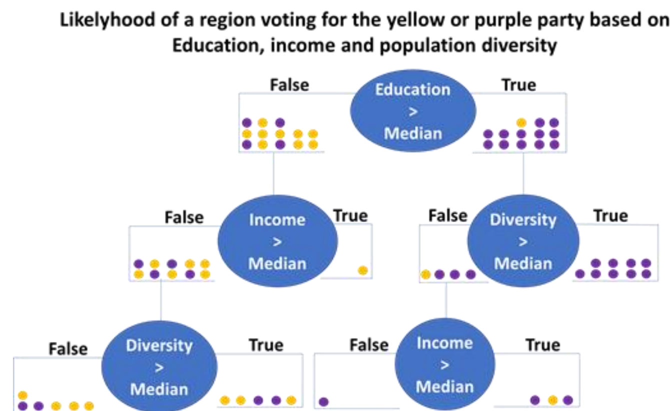

Figure 38 – *Illustration of a decision tree that tries to determine the probability of a state voting for a yellow (represented as False and located on the left side) or a purple party (represented as True and located on the right side).*

In mathematical terms, the tree decides which outcome is more likely by calculating the information gain, determined by the Gini index or entropy. Gini determines the information gain through expression (4), and entropy determines the information gain through expression (5). In both expressions, p_i represents the probability of class i occurring, and with binary outcomes, there are only two classes. Their behavior is also similar because when they reach the maximum value, it means that the probability of each outcome is the same, and there is little information gain. When a node reaches its minimum value, the node is pure and there is no need for further nodes.

$$Gini = \sum_{i=0}^{n} p_i^2 \qquad (4)$$

$$Entropy = - \sum_{i=0}^{n} p_i \, log_2 p_i \qquad (5)$$

The Gini index and entropy generally produce similar results on the same dataset and can be used to determine the root of the decision tree. Since choosing the tree's root can impact the prediction of the outcome, it is more common to use random forests, which are multiple decision trees trained based on the available data. Since various trees are created, several permutations of the variables in the dataset are created for each tree. After complete the training, each tree votes for an outcome (true or false), and the algorithm chooses the outcome with the majority of votes. Due to their nature, random forests are beneficial in problems with two possible outcomes such as spam classification and other security-related issues.

XGBoost is seen as an evolution to Random forests and decision trees and utilizes the gradient descent algorithm to minimize prediction errors. XGBoost (XGB) is similar to Gradient Boosting at its core. It builds decision trees sequentially, approximating its predictions to the observed values by reducing the residuals at each step. The most important differences are regularization parameters added to stimulate pruning the trees and prevent overfitting and software optimizations to maximize computing efficiency and mathematical simplifications to reduce calculation complexity while retaining accuracy. The resulting model is easy to implement, quick to train and achieves very good accuracy in a wide range of problems [313].

An XGBoost classifier starts with a base prediction of 0.5. It then calculates each observation's residuals (observed value - predicted probability) and starts to build the first tree. When building the tree, XGBoost will begin by adding every observation's residual to a single leaf and calculating the similarity score of the leaf using expression 6, where r_i is the residual of observation i, p_i is the last predicted probability of observation i (for the first tree, p_i will always be 0.5), and λ is a regularization parameter.

$$Similarity\ Score = \frac{\left(\sum_{i=0}^{n} r_i\right)^2}{\sum_{i=0}^{n}\left(p_i \times (1-p_i)\right)+\lambda} \qquad (6)$$

The classifier will then try to split the leaf using the data features. For continuous features, XGB will test splitting based on quantiles, and calculate the gain of splitting at each quantile. The feature with the highest gain is chosen and XGB continues trying to split with the other features. Gain is calculated using expression 7, where root similarity

is the original leaf similarity containing all the residuals, $leaf_1$ and $leaf_2$ are the resulting leaves of the split.

$$Gain = simLeaf_1 + simLeaf_2 - simRoot \quad (7)$$

Once the tree is built, XGB will attempt to prune it bottom-up using the γ parameter, eliminating splits where the resulting gain is smaller than γ. Pruning will stop when the gain is larger than γ even if there are splits further up with smaller gains. The output value for each of the tree's leaves is then calculated using Equation 8, where r_i is the residual of observation *i* for each observation in the leaf, p_i is the last predicted probability of observation i (for the first tree it is 0.5), and λ is the regularization parameter. Note that this equation is very similar to equation 6, and both of them will output large values for leaves containing very similar values, while penalizing leaves where residuals are too different, which makes sense as similar residuals indicate good confidence in the tree output.

$$Output\ value = \frac{\sum_{i=0}^{n} r_i}{\sum_{i=0}^{n}(p_i \times (1-p_i)) + \lambda} \quad (8)$$

XGB will keep building trees this way until new trees fail to improve prediction. The final prediction is a log(odds) of the sum of each tree output value times the learning rate (η parameter) for every tree m, as seen in equation 9. The initial prediction of 0.5 does not get added because the log(odds) of a 0.5 probability is 0. The final log(odds) value is then converted into a probability using equation 10. A deeper analysis of XGBoost mathematical formulation can be found in the original paper [314].

$$\log(odds) = \sum_{m=0}^{M} \eta \times outputvalue_m \quad (9)$$

$$Probability = \frac{e^{\log(odds)}}{1+e^{\log(odds)}} \quad (10)$$

8.3 Experimentation scenario and result analysis

This section presents the stages taken to identify replication attacks through XGBoost and analyzes the testing data results. XGBoost was chosen to solve such an issue because they performed well in Kaggle competitions, are easy to train, produce accurate results in labeled datasets, and are consistent across various scenarios. The following steps were taken to train the model: *i*) Obtain or produce a dataset, *ii*) clean dataset, *iii*) grid search to find the best training parameters.

- **Obtain or produce a dataset**: This involves gathering large amounts of data, either through the usage of an existing dataset or by data generation. In this study, IoT-23 was used [296]. It is a publicly labeled dataset containing benign and malicious IoT network traffic captured from 2018 to 2019 [297]. he dataset labels several threats like Okiru, Torii, Mirai, port scanning DDos, C&C, and heartbeat. More details regarding the dataset can be found in Subsection 8.1.3.

- **Clean dataset**: At this step, gathered data is analyzed to decide which data will be used and generally also involves data normalization (also known as a feature normalization in machine learning). Data normalization is widespread in machine learning, especially when numerical data is used and it consists on rescaling attributes to a given range (generally from 0 to 1). Without rescaling numerical data, the model might not correctly estimate the importance of the numerical values. One of the advantages of XGBoost is that numerical data does not need to be normalized. Categorical features were one-hot encoded, which is a technique similar to normalization but for labeled data, and it is based on transforming every category into a boolean variable, where 1 (one) represents True, and 0 (zero) False. The following features were one-hot encoded: network protocol (tcp, icmp, udp), application-layer protocol (DNS, SSL, SSH, DHCP, HTTP, IRC), and connection state, which has 13 (thirteen) categories

- **Grid search to find the best training parameters**: When training machine learning models, a crucial aspect is choosing the training hyperparameters set before training. Grid search is a technique that searches for the best parameter values of a chosen model. For XG Boost, the hyperparameters on the grid search were η, γ, λ, depth, scale, and rounding. Depth refers to the number of questions made in each decision tree. η is the learning rate to scale each tree's output value. γ and λ are regularization parameters to stimulate tree pruning and avoid overfitting. Scale gives more weight to the less common observation value, to compensate for skewed data. Rounding is the threshold used when rounding the percentage output by the model to a final label. The grid search results are presented in Table 5. Results below rank 3 are omitted and reveal the difference between the ranked 1 and ranked 3 parametrization was minimal. Overall, variations of the depth = 6 occupy most of the top positions.

Table 5 – Grid search parameters to find the optimal training parameters considering η, γ, λ, depth, scale, rounding, accuracy, and f1 score.

η	γ	λ	depth	scale	rounding	accuracy	F1	Rank
0.1	0	1	6	0.3	0.3	0.883	0.937	1
0.1	0	0	6	0.3	0.5	0.882	0.937	2
0.1	0	0	6	0.3	0.3	0.882	0.937	2
0.5	0	1	6	0.3	0.3	0.882	0.937	2
0.1	1	1	6	1.0	0.3	0.880	0.936	3
0.1	1	1	6	1.0	0.3	0.880	0.936	3
0.1	0	1	6	1.0	0.3	0.880	0.936	3
0.1	1	1	15	1.0	0.3	0.880	0.936	3
0.1	1	1	15	1.0	0.3	0.880	0.936	3
0.1	1	0	15	1.0	0.3	0.880	0.936	3

8.3.1 Result analysis

The number one ranked parametrization was used to evaluate the model ($\eta = 0.1$, $\gamma = 0$, $\lambda = 1$, depth = 6, scale = 0.3, and rounding = 0.3). The trained model presents 93.6% accuracy, the precision of 93.7%, a recall score of 99.9%, and an F1 score of 96.7% in the data used in the experiments. The results are summarized in the confusion matrix presented in Figure 39 and shows the model detects more false positives than false negatives, which is preferable in security applications. With a higher number of false positives, compromised devices are more likely to be detected. If the model accused more false negatives than positives, several infected devices would remain undetected. The hidden cost of a high number of false positives is that they require further investigation from a network administrator.

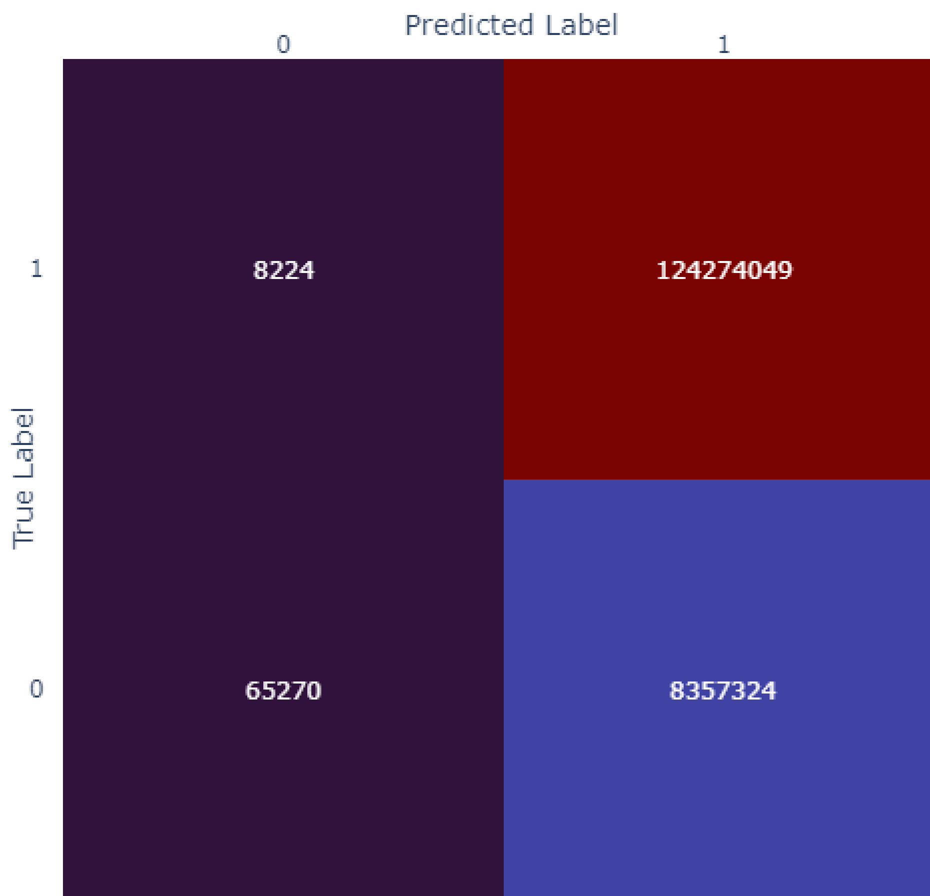

Figure 39 – *Confusion matrix results of the test data – 1 means attack.*

When identifying if the request was from an attacker, the trained model attributed more importance to the following features: Duration of the request (2875 splits), orig_ip_bytes (898 splits), orig_ip_pkts (600 splits), Orig_Bytes (566), Resp_bytes (258), resp_pkts (243), conn_state_rej (205), proto_tcp (200), conn_state_S0 (155), and resp_IP_bytes (145). The feature importance is displayed in Figure 40. Features with a higher number of splits are more relevant to the model.

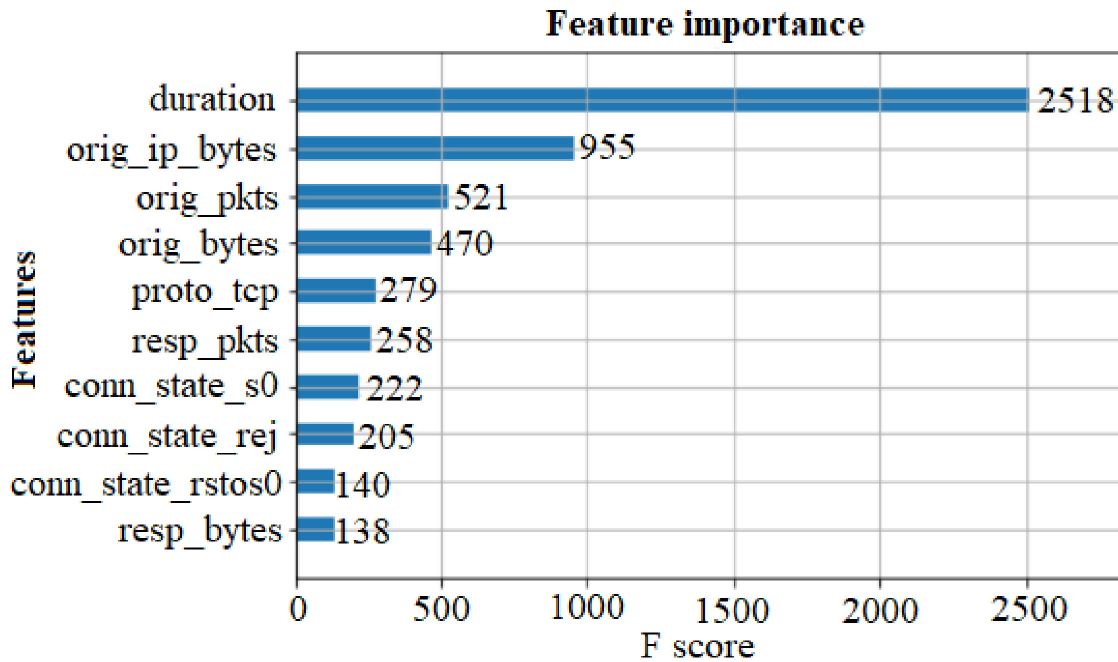

Figure 40 – *Feature importance determined by the trained model.*

The duration of the request determines how long the request took to be fulfilled, so it makes sense that the model determines that requests with a higher duration originate from an attacker in a distant region, thus giving higher importance for the duration. It is hard to determine the reasons for the model attributing importance to the other features because, to humans, little information can be extracted from them. Orig_ip_bytes is the number of bytes from the originator (external host) IP. Orig_pkts packets is the number of packets that came from the external host. Orig_Bytes refers to the payload bytes of the external host. Resp_Bytes refers to the payload bytes sent by the server. Resp_pkts is the number of packets that came from the server, conn_state_rej means that connection was attempted by the originator and rejected by the server. Proto_tcp means that the used protocol was TCP. Conn_state_s0 means that the connection attempt was seen by the server and received no reply. Rest_ip_bytes is the number of bytes from the server (external host) IP.

8.4 Summary

Security is a crucial and delicate topic often neglected in IoT environments because IoT devices generally present numerous constraints such as limited memory, processing power, and battery. This lack of security attracts various attackers that easily exploit device vulnerabilities. The chapter presented an overview of the most common

threats to IoT platforms and devices as well as the best practices to mitigate these threats. The study focused on detecting replication attacks where attackers obtain device credentials to disrupt the IoT environment. The replication attack is hard to detect and could have severe consequences to the entire network and should be addressed as presented in this chapter.

The study used XGBoost to detect the replication attack and the public dataset IoT-23 to train and perform the experiments to evaluate the proposed model. IoT-23 contains device data from infected and healthy devices recorded from 2018 to 2019. The obtained results were extremely promising because, after the training phase, the system presented an accuracy of 93.6%, a recall score of 0.99, a precision of 93.6%, and an F1 score of 96.7%. The trained model also detected more false positives than negatives, which is a good result in security solutions because otherwise, infected devices might not be detected. Also, the trained model could be integrated into an IoT middleware solution and further and block access devices that were flagged as suspicious, further increasing the security of IoT environments.

9 Conclusion and Future Works

Middleware is very important in IoT environments, mainly because of its role of enabling the communication among devices, users, and applications. Understanding the main issues regarding security and usability of the most popular solutions enables researchers and developers to keep improving such vital software. Constructing In.IoT was a challenging task that derived from difficulties of modifying Sitewhere. In the beginning of the project, Sitewhere was used as a Middleware solution because previous studies demonstrated that it is a solution with good performance. Then, the most relevant application layer protocols for IoT were identified so they could be added to the list of supported protocols, but this task turned out to be too complex. For this reason, MiddleBridge was created, enabling Sitewhere and any middleware solution to support more protocols.

Fixed the protocol issue, it was time to improve the security functionalities of Sitewhere, but modifying the solution turned out to be far more difficult than expected and the team decided that it would be easier to build a solution from scratch. The new solution could then be compliant with the middleware reference architecture, combine other functionalities and support multiple protocols. Since the solution was built from scratch, and performance was a crucial factor when evaluating the solution, the group decided to optimize every possible aspect of the solution without compromising security. For this reason, the performance of the most popular programming languages and their respective frameworks was analyzed, reaching the conclusion that Java is the most robust programming language.

Since Java demonstrated to be the most robust programming language, it was used for the core modules of In.IoT. The first version of In.IoT only supported HTTP as an application layer protocol natively and MiddleBridge had to be used for the other protocols. MQTT and CoAP support was added in later versions, and currently

MiddleBridge used along with In.IoT when support for the DDS and Websockets protocols are needed.

In.IoT is a flexible middleware solution that can be used across various IoT scenarios, but like any middleware, for improved data visualization on specific scenarios, third-party applications that make use of the API must be developed. With this in mind, OLP was developed, a simple low-code development platform that allows users with little programming experience to develop simple web-based applications through visual representations.

The last work on the research project consisted of detecting the occurrence of replication attacks where an attacker obtains device credentials, using it to generate false data and disturb the IoT environment. This type of attack is very difficult to detect and since IoT middleware are located on powerful servers, these servers could use the additional resources to detect such an attack. In this sense, a machine learning model was trained using XGBoost to detect the occurrence of such an attack

9.1 Learned lessons

Researchers and software developers interested in evaluating or developing IoT middleware solutions, low-code platforms, or creating a predictive machine learning model can benefit from the findings of this research study and are detailed in this subsection.

The tradeoff between security and performance can be minor if appropriately done: Whenever security increases, performance decreases because the added security generally consists of more verifications. However, the added security can increase performance if certain characteristics of the chosen security technique are exploited (like in In.IoT architecture). Also, since IoT middleware is located in a server, the tradeoff should be minimal with proper security.

Developers should use tools they are familiar with: Deciding which tools to use is challenging because the front-end, which will interact with end-users, might be built in a different programming language from the backend that will process the requests. The suggestion is for developers to use tools they are familiar with because building complex software such as an IoT middleware or a low-code platform is already

a challenging task that will demand much effort. Then, adding the learning curve of a new programming language will only make the project more difficult.

The architecture should be flexible and support future modifications: A good architecture reflects its software requirements. With IoT middleware and low-code, it is crucial that the architecture is flexible and can be applied to various use-cases. To verify the scalability and flexibility, the developers can discuss which elements of the architecture could be combined or removed. This aspect related to the architecture is especially valuable in building a low-code platform because since it is software used to build other software, scaling will be an issue soon if the architecture is not flexible.

Operator precedence and blank spaces can be the source of various bugs: Operator precedence determines how expressions are evaluated. Therefore, it should be well-defined and tested. Although low-code is a mostly visual, textual representations are still used in the code generator and in expressions typed by the end-user. The code generator uses a textual version of the visual language to implement functionalities and the user can type expressions in text boxes if they wish to. In both situations, operators can be used and their precedence must be considered. In early development, a mistake while coding the parsing of "XOR" and "AND" ("AND" was attributed a higher precedence), which could have caused several bugs later on. Nowadays, most programming languages have the same operator precedence, but developers should verify whether an operator precedence is well-defined to minimize bugs on the generated code. Also, end-users can ident their written code in various ways and the platform should support the most common blank spaces such as "Tab" and "space".

Write many test programs to test the platform functionalities: Developing tools to debug and automate testing of the platform functionalities should be a top priority because it is easy to break code when the application also generates source code. Without such tools, several bugs will be introduced with each newly added functionality. Test programs should also be used in IoT platform development.

9.2 Final remarks

Throughout this thesis, an updated study on IoT middleware was presented, and several improvements were made on the topic. The thesis introduced the motivation and

delimited the research topic in the first chapter, describing the objectives and its main contributions.

Chapter 2 provided an invaluable background on the IoT topic. The chapter begins an overview regarding devices classification according to processing and networking capabilities. Next, the chapter provided a summary of the most common ways of connecting to the Internet in the IoT, followed by an overview of message exchange patterns in communications over the Internet, the topic is important because IoT communications mostly rely on the PubSub paradigm. Next, the chapter summarized the difficulty of enforcing a global standard for objects to interact with each other on the IoT, as well as the most popular application layer protocols for IoT (HTTP, CoAP, and MQTT), and despite HTTP inefficiency in IoT applications it is still widely used. Next, the need for an IoT middleware was introduced and an overview regarding a reference architecture IoT Middleware was provided, this is crucial because the reference architecture is the pillar for the entire research. Finally, the chapter introduced the authorization mechanisms for IoT middleware and explains the main issues with the most relevant open-source middleware solutions, providing tangible examples regarding the most popular open-source middleware solutions

Chapter 3 examined the difficulty of determining the best global middleware solution in comparative studies. The chapter introduced the most popular qualitative and quantitative evaluation metrics for the IoT middleware topic, as well as some of the studies that are available in the literature and their main conclusions. The evaluation metrics are important because throughout the thesis, several conclusions are supported by these metrics. Moreover, the few quantitative comparisons that are available in the literature can only determine the best solution in separated given categories. Then, the chapter introduced PROMETHEE, a multi-criteria decision making method that can be used to combine distinct and at first sight incompatible comparison metrics. With MCDM methods, a best global solution can be found by attributing weights for each metric. Then, the chapter complemented the conclusions of quantitative comparison study that is available in the literature through a comparison of five middleware solutions across five different scenarios. The study also confirmed that the best solution depends on which criteria are prioritized in a given scenario. The outcome was analyzed in detail and it was concluded that MCDMs are useful when choosing the best middleware platform to deploy in a given IoT solution. Orion (a Fiware project), InatelPlat, and Sitewhere are the platforms that performed better in the study.

Chapter 4 studied the scenario where an IoT device purchased by an end-user is incompatible with at none of the application layer protocols supported by the middleware, and such gadget does not achieve its potential regarding functionalities. Next, the chapter presented an overview of application layer gateways (also called application layer bridges), and the some of their existing implementations for IoT environments. Then, the chapter proposed MiddleBridge, that translates CoAP, MQTT, DDS, and Websockets messages into HTTP. MiddleBridge can be deployed on any computer with Java virtual machine because all servers are embedded in its code, enabling IoT gadgets to transmit data to any REST endpoint seamlessly. With the proposed approach, devices can send a smaller message to an intermediary (MiddleBridge), which restructures it and forwards to a middleware, reducing the time that a device spends transmitting. The created graphical user interface allows users to configure messages conversion and forwarding in runtime. The efficiency of such approach was evaluated through the packet size and response times considering the data sent to Orion context broker (a Fiware project). Results showed that packet size that is sent by an IoT device through MiddleBridge is 17 times smaller than sending a straight HTTP request to the server and significantly reduces the transmission time.

Chapter 5 analyzed the impact of the underlying programming language on the performance of middleware solutions. The chapter introduced previous performance evaluation studies regarding programming languages, but most of them consist on algorithms that do not capture the essence of what is important for IoT middleware, the communication with databases and the capability to handle simultaneous web requests. Next, the chapter introduced the most popular programming languages and frameworks to build REST APIs, this was based on the combination of the most popular programming languages and most popular frameworks on Stack Overflow (two distinct reports from the year 2019). The chapter then analyzed the performance of Express (Javascript), Spring (Java), and Flask (Python) frameworks. The chapter concluded that Java is the most robust overall, displaying similar behavior independently of the number of parameters. Python, despite its seemingly high throughput, has enormous failure rates and is not adequate for heavy loads. If performance is the most important metric, then Javascript should be the language of choice. However, for data-heavy applications, or situations where robustness is the objective, Java is the best choice.

Chapter 6 introduced In.IoT, a new middleware platform for IoT, that addresses the concerns identified through the thesis. In.IoT is based on the microservices architecture, being scalable, secure, and innovative. The chapter introduced an overview to the microservices architecture, showing the distinctions between the microservices and SOA architecture. Next, the chapter reviewed the most popular authorization mechanisms, concluding that client tokens could be useful for IoT applications. Next, the chapter presented In.IoT architecture, that can be replicated by any new or even existing solution presenting details regarding its data storage, security, and other operational aspects to increase performance. Then, the chapter discussed aspects relative to In.IoTs construction, and its features relative to the middleware reference architecture. Finally, the chapter presented a performance evaluation study in comparison with the most promising solutions available in the literature and the results obtained by the proposed solution are extremely promising. In.IoT is evaluated, demonstrated, validated, and it is ready and available for use.

Chapter 7 introduced OLP, a low-code platform that allow users with little coding experience to develop applications through visual representations that are transpiled into source code. With OLP, users can build custom applications for In.IoT or any other middleware by persisting and consulting data through the REST APIs. The chapter presented the low-code concept as well as the differences between low-code, no-code and the traditional approaches. Next, the requirements and architecture of this type of application was showcased. Finally, the chapter presented the details regarding the development and the application was demonstrated.

Chapter 8 introduced a security application based on machine learning and detects the occurrences of the replication attack, where an attacker obtains device credentials and disrupts the IoT environment. The chapter introduced the most common threats to IoT environment and how to mitigate them. Then, an overview regarding decision trees, random forest, and XGBoost was presented. Finally, the dataset that was used to train the ML model was presented, the experimentation scenario was showcased and the obtained results were displayed.

It is concluded that all the goals were successfully achieved throughout the thesis.

9.3 Future works

To complement this research work, the following topics are suggested for future research directions:

- Future work regarding **Chapter 3** should improve the scoring system (ranking in each criterion the middleware solutions from 1 to 5) that was used because, although efficient, solutions can be penalized when the difference among them is minimal in one criteria. Such behavior was verified when analyzing the packet loss, in which the difference between Orion and Sitewhere was minimal. However, the overall difference was significant because other solutions performed better in some sub-categories of packet loss. The improvement considered is based on the difference between each criterion's minimum and maximum values, adjusted in a continuous range from 0 to 5 (instead of ranking each one from 1 to 5).
- Regarding MiddleBridge, presented in **Chapter 4**, an evolution could focus on supporting bi-directional conversion among the multiple protocols and support other IoT protocols that might be proposed, further extending the flexibility of the solution. The translation latency for each protocol could also be analyzed as well as the impact the shorter transmissions and reduced packet size can represent for the power consumption in IoT devices. Also, a JMeter plugin that supports DDS should be developed to evaluate the efficiency and scalability of the DDS protocol in MiddleBridge. In the future, solutions like MiddleBridge can even optimize the packet size of requests made in the same protocol, especially in “heavy” protocols such as HTTP. Furthermore, an evolution of the proposed solution could support bi-directional conversion among the multiple protocols and support other IoT protocols that might be proposed further, extending the solution's flexibility.
- Regarding the impact of the underlying programming languages study presented in **Chapter 5**, future work should include more programming languages such as Go and C++. Also, different test cases, such as: message protocols handling performance, streaming of data for user analysis, and simultaneous overload in different types of operations. Furthermore, the usage of different databases (both SQL and NoSQL) is also necessary to isolate the MongoDB libraries' possible effects for each language.

- Regarding the In.IoT middleware platform presented in **Chapter 6**, future work can focus on studying the impact of In.IoT in devices energy consumption. Furthermore, improving some security aspects that are currently not addressed by In.IoT could be relevant, such as identifying and alerting a human user when device credentials are compromised. Another contribution could expand the number of supported application layer protocols and could include DDS or another upcoming protocol. Another useful study could consist of implementing In.IoT architectural requirements and recommendations with different programming languages to investigate their impact on the solution's performance. Such a study could use programming languages, such as Go, C++, and JavaScript runtime builds such as NodeJS to compare the performance relative to this initial Java implementation and verify which language suits better for the solution.
- Regarding the OLP presented in **Chapter 7**, future works can improve usability and support automatic deployment after generating the project artifacts. The project can also provide native data storage instead of relying on external REST APIs and allow users to write HTML code. Furthermore, support for other mediums, such as mobile applications, could vastly improve the solutions' utility. Perhaps the biggest issue with the low-code approach is that since the platform generates source code, when the code contains vulnerabilities, every application that uses the platform inherits it.
- Regarding the detection of compromised devices in **Chapter 8**, future works could focus on evaluating the system's efficiency with other datasets to verify the flexibility of the trained model. Moreover, the efficiency of XGBoost in this particular problem could be compared with Random forests or a GAN (Generative Adversarial Network) where the model trains itself based on a dataset. GANs considers two neural networks and their training is different in the sense that, during training, one neural network is a forger that tries to deceive a fraud detector (which is the other neural network). In the context of this work, the forger would attempt to replicate requests that are similar to a legitimate request and the fraud detector would attempt to detect the fraudulent request.

References

[1] P. P. Ray, “A survey on Internet of Things architectures,” *Journal of King Saud University - Computer and Information Sciences*, vol. 30, no. 3, pp. 291–319, Jul. 2018.

[2] Carlo Puliafito, Enzo Mingozzi, Francesco Longo, Antonio Puliafito, and Omer Rana, “Fog Computing for the Internet of Things: A Survey,” *ACM Transactions on Internet Technology*, vol. 19, no. 2, pp. 1–41, Apr. 2019.

[3] L. Atzori*, J. L. Bellido, R. Bolla, G. Genovese, A. Iera, A. Jara, C. Lombardo, G. Morabito*, “SDN&NFV contribution to IoT objects virtualization,” *Computer Networks*, vol. 149, pp. 200–212, Feb. 2019.

[4] Yazdan A. Qadri, Ali Nauman, Yousaf Bin Zikria, Athanasios V. Vasilakos, and Sung W. Kim, “The Future of Healthcare Internet of Things: A Survey of Emerging Technologies,” *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1121–1167, Feb. 2020.

[5] Lei Xu, Lin Chen, Zhimin Gao, Xinxin Fan, Taewon Suh, and Weidong Shi, “DIoTA: Decentralized-Ledger-Based Framework for Data Authenticity Protection in IoT Systems,” *IEEE Network*, vol. 34, no. 1, pp. 38–46, Jan. 2020.

[6] Hamid Tahaei, Firdaus Afifi, Adeleh Asemi, Faiz Zaki, and Nor B. Anuar, “The rise of traffic classification in IoT networks: A survey,” *Journal of Network and Computer Applications*, vol. 154, Mar. 2020, p. 102538, Mar. 2020.

[7] Kewei Sha, T. A. Yang, Wei Wei, and Sadegh Davari, “A survey of edge computing-based designs for IoT security,” *Digital Communications and Networks*, vol. 6, no. 2, pp. 195–202, May 2020.

[8] Libo Jiao, Yulei Wu, Jiaqing Dong, and Zexun Jiang, “Toward Optimal Resource Scheduling for Internet of Things Under Imperfect CSI,” *IEEE Internet Things Journal*, vol. 7, no. 3, pp. 1572–1581, Mar. 2020.

[9] Jun Zhou, Zhenfu Cao, Xiaolei Dong, and Athanasios V. Vasilakos, “Security and Privacy for Cloud-Based IoT: Challenges,” *IEEE Communications Magazine*, vol. 55, no. 1, pp. 26–33, Jan. 2017.

[10] Zhongxiang Wei, Christos Masouros, Fan Liu, Symeon Chatzinotas, and Bjorn Ottersten, “Energy- and Cost-Efficient Physical Layer Security in the Era of IoT: The Role of Interference,” *IEEE Communications Magazine*, vol. 58, no. 4, pp. 81–87, Apr. 2020.

[11] Damian Arellanes and Kung-Kiu Lau, “Evaluating IoT service composition mechanisms for the scalability of IoT systems,” *Future Generation Computer Systems*, vol. 108, pp. 827–848, Jul. 2020.

[12] G. Aloi, G. Caliciuri, G. Fortino, R. Gravina, P. Pace, W. Russo, C. Savaglio, “Enabling IoT interoperability through opportunistic smartphone-based mobile gateways,” *Journal of Network and Computer Applications*, vol. 81, pp. 74–84, Mar. 2017.

[13] Aneesh M. Koya and P. P. Deepthi, “Plug and play self-configurable IoT gateway node

for telemonitoring of ECG," *Comput. Biol. Med.*, vol. 112, p. 103359, Sep. 2019.

[14] Tshiamo Sigwele, Yim F. Hu, Muhammad Ali, Jiachen Hou, Misfa Susanto, and Helmy Fitriawan, "Intelligent and Energy Efficient Mobile Smartphone Gateway for Healthcare Smart Devices Based on 5G," *2018 IEEE Global Communications Conference (GLOBECOM 2018)*, Abu Dhabi, United Arab Emirates, 9-13 Dec. 2018, pp. 1-7.

[15] Mauro A. A. da Cruz, Joel J. P. C. Rodrigues, Jalal Al-Muhtadi, Valery V. Korotaev, and Victor H. C. de Albuquerque, "A Reference Model for Internet of Things Middleware," *IEEE Internet Things Journal*, vol. 5, no. 2, pp. 871–883, Apr. 2018.

[16] Mauro A. A. da Cruz, Joel J. P. C. Rodrigues, Arun K. Sangaiah, Jalal Al-Muhtadi, and Valery Korotaev, "Performance evaluation of IoT middleware," *Journal of Network and Computer Applications*, vol. 109, pp. 53–65, May 2018.

[17] Wayne D. Hoyer, Mirja Kroschke, Bernd Schmitt, Karsten Kraume, and V. Shankar, "Transforming the Customer Experience Through New Technologies," *Journal of Interactive Marketing*, vol. 51, no. 3, pp. 57–71, Aug. 2020.

[18] Sharu Bansal and D. Kumar, "IoT Ecosystem: A Survey on Devices, Gateways, Operating Systems, Middleware and Communication," *International Journal of Wireless Information Networks*, vol. 27, no. 3, pp. 340–364, Feb. 2020.

[19] Shivangi Vashi, Jyotsnamayee Ram, Janit Modi, Saurav Verma, and Chetana Prakash, "Internet of Things (IoT): A vision, architectural elements, and security issues," *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)*, Palladam, India, 10-11 Feb. 2017, pp. 492-496.

[20] Prasanna K. Illa and Nikhil Padhi, "Practical Guide to Smart Factory Transition Using IoT, Big Data and Edge Analytics," *IEEE Access*, vol. 6, pp. 55162–55170, 2018.

[21] Jin-Yong Yu and Young-Gab Kim, "Analysis of IoT Platform Security: A Survey," in *2019 International Conference on Platform Technology and Service (PlatCon)*, Jeju, South Korea, 28-30 Jan. 2019, pp. 1–5.

[22] Debajyoti Misra, Gautam Das, and D. Das, "Review on Internet of Things (IoT): Making the World Smart," in *Advances in Communication, Devices and Networking*, Springer Sangapore, 2018, pp. 827–836.

[23] Cristina Paniagua and Jerker Delsing, "Industrial Frameworks for Internet of Things: A Survey," *IEEE Systems Journal*, vol. 15, no. 1, pp. 1149–1159, Mar. 2021.

[24] Qin Wang, Xinqi Zhu, Yiyang Ni, Li Gu, and Hongbo Zhu, "Blockchain for the IoT and industrial IoT: A review," *Internet of Things*, vol. 10, p. 100081, Jun. 2020.

[25] W. Z. Khan, M. H. Rehman, H. M. Zangoti, M. K. Afzal, N. Armi, and K. Salah, "Industrial internet of things: Recent advances, enabling technologies and open challenges," *Computers & Electrical Engineering*, vol. 81, p. 106522, Jan. 2020.

[26] H. Ali, N. H. Eldrup, F. Normann, V. Andersson, R. Skagestad, A. Mathisen, and L. Erik, "Cost estimation of heat recovery networks for utilization of industrial excess heat for carbon dioxide absorption," *International Journal of Greenhouse Gas Control*, vol. 74, pp. 219–228, Jul. 2018.

[27] S. Verma, Y. Kawamoto, Z. M. Fadlullah, H. Nishiyama, and N. Kato, “A Survey on Network Methodologies for Real-Time Analytics of Massive IoT Data and Open Research Issues,” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1457–1477, 3rd Quart. 2017.

[28] B.-Y. Ooi and S. Shirmohammadi, “The potential of IoT for instrumentation and measurement,” *IEEE Instrumentation & Measurement Magazine*, vol. 23, no. 3, pp. 21–26, May 2020.

[29] A. Riahi Sfar, Y. Challal, P. Moyal, and E. Natalizio, “A Game Theoretic Approach for Privacy Preserving Model in IoT-Based Transportation,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 12, pp. 4405–4414, Dec. 2019.

[30] M. Babar, M. U. Tariq, and M. A. Jan, “Secure and resilient demand side management engine using machine learning for IoT-enabled smart grid,” *Sustainable Cities and Society*, vol. 62, p. 102370, Nov. 2020.

[31] Shabrukh K. Kasi, Mumraiz K. Kasi, Kamran Ali, Mohsin Raza, Hifza Afzal, Aboubaker Lasabae, Bushra Naeem, Saif ul Islam, and Joel J. P. C. Rodrigues, “Heuristic Edge Server Placement in Industrial Internet of Things and Cellular Networks,” *IEEE Internet of Things Journal*, pp. 1-1, 2020.

[32] M. S. Farooq, S. Riaz, A. Abid, T. Umer, and Y. Bin Zikria, “Role of IoT Technology in Agriculture: A Systematic Literature Review,” *Electronics*, vol. 9, no. 2, p. 319, Feb. 2020.

[33] M. A. Akkaş, R. SOKULLU, and H. Ertürk Çetin, “Healthcare and patient monitoring using IoT,” *Internet of Things*, vol. 11, p. 100173, Sep. 2020.

[34] R. Sokullu, M. A. Akkaş, and E. Demir, “IoT supported smart home for the elderly,” *Internet of Things*, vol. 11, p. 100239, Sep. 2020.

[35] L. F. Luque-Vega, M. A. Carlos-Mancilla, V. G. Payán-Quiñónez, and E. Lopez-Neri, “Smart cities oriented project planning and evaluation methodology driven by citizen perception-IoT smart mobility case,” *Sustainability*, vol. 12, no. 17, p. 7088, 2020.

[36] International Telecommunication Union, “ITU-T Y. 4460 - Architectural Reference Model of Devices for IoT Applications,” Geneva, Switzerland: International Telecommunications Union, 2019.

[37] Mauro A. A. Da Cruz, Guilherme A. B. Marcondes, Joel J. P. C. Rodrigues, Pascal Lorenz, and Plácido R. Pinheiro, “Performance Evaluation of IoT Middleware through Multicriteria Decision-Making,” in *2018 IEEE Global Communications Conference (GLOBECOM)*, Abu Dhabi, United Arab Emirates, 9-13 Dec. 2018, pp. 1–5.

[38] Mauro A. A. da Cruz, Joel J. P. C. Rodrigues, Pascal Lorenz, Petar Solic, Jalal Al-Muhtadi, and Victor H. C. Albuquerque, “A proposal for bridging application layer protocols to HTTP on IoT solutions,” *Future Generation Computer Systems*, vol. 97, pp. 145–152, Aug. 2019.

[39] Lucas R. Abbade, Mauro A. A. da Cruz, Joel J. P. C. Rodrigues, Pascal Lorenz, Ricardo A. L. Rabelo, and Jalal Al-Muhtadi, “Performance comparison of programming languages for Internet of Things middleware,” *Transactions on Emerging Telecommunications*

Technologies, vol. 31, no. 12, e3891, Dec. 2020.

[40] Mauro A. A. da Cruz, Joel J. P. C. Rodrigues, Pascal Lorenz, V. Korotaev, and V. H. C. de Albuquerque, “In.IoT—A new Middleware for Internet of Things,” *IEEE Internet Things Journal*, vol. 8, no. 10, pp. 7902–7911, May 2021.

[41] I. Sommerville, *Software Engineering*, 10th ed., Harlow, U.K.: Addison-Wesley, 2010.

[42] D. J. Langley, J. van Doorn, I. C. L. Ng, S. Stieglitz, A. Lazovik, and A. Boonstra, “The Internet of Everything: Smart things and their impact on business models,” *Journal of Business Research*, vol. 122, pp. 853–863, Jan. 2021.

[43] M. H. Miraz, M. Ali, P. S. Excell, and R. Picking, “Internet of Nano-Things, things and everything: Future growth trends,” *Future Internet*, vol. 10, no. 8, P. 68, Jul. 2018.

[44] J. Wang, M. K. Lim, C. Wang, and M.-L. Tseng, “The evolution of the Internet of Things (IoT) over the past 20 years,” *Computers & Industrial Engineering*, vol. 155, p. 107174, May 2021.

[45] D. Valentinetti and F. Flores Muñoz, “Internet of things: Emerging impacts on digital reporting,” *Journal of Business Research*, pp. 1-1, 2021.

[46] C. Wei, “5G-oriented IoT coverage enhancement and physical education resource management,” *Microprocessors and Microsystems*, vol. 80, pp. 103346, Feb. 2021.

[47] F. John Dian, R. Vahidnia, and A. Rahmati, “Wearables and the Internet of Things (IoT), Applications, Opportunities, and Challenges: A Survey,” *IEEE Access*, vol. 8, pp. 69200–69211, Apr. 2020.

[48] C. Qiu, F. Wu, C. Lee, and M. R. Yuce, “Self-powered control interface based on Gray code with hybrid triboelectric and photovoltaics energy harvesting for IoT smart home and access control applications,” *Nano Energy*, vol. 70, p. 104456, Apr. 2020.

[49] M. Condoluci, T. Mahmoodi, M. A. Lema, and M. Dohler, “5G IoT industry verticals and network requirements,” in *Powering Internet Things With 5G Networks*, IGI Global, pp. 148–175, 2018.

[50] A. A. Mutlag, M. K. Abd Ghani, N. Arunkumar, M. A. Mohammed, and O. Mohd, “Enabling technologies for fog computing in healthcare IoT systems,” *Future Generation Computer Systems*, vol. 90. pp. 62–78, Jan. 2019.

[51] S. Bresciani, A. Ferraris, and M. Del Giudice, “The management of organizational ambidexterity through alliances in a new context of analysis: Internet of Things (IoT) smart city projects,” *Technological Forecasting and Social Change*, vol. 136, pp. 331–338, Nov. 2018.

[52] F. Arena, G. Pau, and A. Severino, “An Overview on the Current Status and Future Perspectives of Smart Cars,” *Infrastructures*, vol. 5, no. 7, p. 53, Jun. 2020.

[53] A. M. Alberti, “A conceptual-driven survey on future internet requirements, technologies, and challenges,” *Journal of the Brazilian Computer Society*, vol. 19, pp. 291–311, Mar. 2013.

[54] International Telecommunication Union, *Measuring the Information Society Report 2016*. Geneva, Switzerland, pp. 1-256, 2016.

[55] V. Mulloni and M. Donelli, “Chipless RFID sensors for the internet of things: Challenges and opportunities,” *Sensors*, vol. 20, no. 7, p. 2135, Apr. 2020.

[56] R. Bogdan, A. Tatu, M. M. Crisan-Vida, M. Popa, and L. Stoicu-Tivadar, “A practical experience on the amazon alexa integration in smart offices,” *Sensors*, vol. 21, no. 3, p. 734, Jan. 2021.

[57] V. Kepuska and G. Bohouta, “Next-generation of virtual personal assistants (Microsoft Cortana, Apple Siri, Amazon Alexa and Google Home),” *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC 2018)*, Las Vegas, NV, USA, 8-10 Jan. 2018, pp. 99-103.

[58] A. Kaplan and M. Haenlein, “Siri, Siri, in my hand: Who’s the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence,” *Business Horizons*, vol. 62, no. 1, pp. 15–25, Jan. - Feb. 2019.

[59] H. Isyanto, A. S. Arifin, and M. Suryanegara, “Design and Implementation of IoT-Based Smart Home Voice Commands for disabled people using Google Assistant,” *2020 International Conference on Smart Technology and Applications (ICoSTA 2020)*, Surabaya, Indonesia, 20-20 Feb. 2020, pp. 1-6.

[60] R. K. Ghosh, *Wireless Networking and Mobile Data Management*. Singapore: Springer Singapore, 2017.

[61] Q. Chen and L. Tang, “A wearable blood oxygen saturation monitoring system based on bluetooth low energy technology,” *Computer Communications*, vol. 160, pp. 101–110, Jul. 2020.

[62] R. Casagrande, R. Moraes, C. Montez, A. S. Morales, and L. Rech, “Interference of IEEE 802.11n Networks upon IEEE 802.15.4-Based WBANs: An Experimental Study,” in *2018 IEEE 16th International Conference on Industrial Informatics (INDIN 2018)*, Porto, Portugal, 18-20 Jul. 2018, pp. 388-393.

[63] G. Premsankar, B. Ghaddar, M. Slabicki, and M. Di Francesco, “Optimal Configuration of LoRa Networks in Smart Cities,” *IEEE Transactions on Industrial Informatics*, vol. 16, no. 12, pp. 7243–7254, Dec. 2020.

[64] A. Lavric, A. I. Petrariu, and V. Popa, “Long Range SigFox Communication Protocol Scalability Analysis under Large-Scale, High-Density Conditions,” *IEEE Access*, vol. 7, pp. 35816–35825, Mar. 2019.

[65] L. Zhang and M. Ma, “FKR: An efficient authentication scheme for IEEE 802.11ah networks,” *Computers & Security*, vol. 88, p. 101633, Jan. 2020.

[66] Luiz Oliveira, Joel J. P. C. Rodrigues, Sergei A. Kozlov, Ricardo A. L. Rabêlo, and Vasco Furtado, “Performance assessment of long-range and Sigfox protocols with mobility support,” *International Journal of Communication Systems*, vol. 32, no. 13, e3956, Jul. 2019.

[67] B. Badihi, L. F. Del Carpio, P. Amin, A. Larmo, M. Lopez, and D. Denteneer, “Performance Evaluation of IEEE 802.11ah Actuators,” in *2016 IEEE 83rd Vehicular Technology Conference (VTC Spring)*, Nanjing, China, 15-18 Jul. 2016, pp. 1–5.

[68] A. Šljivo, D. Kerkhove, L. Tian, J. Famaey, A. Munteanu, I. Moerman, J. Hoebeke, and E.

Poorter, “Performance evaluation of IEEE 802.11ah networks with high-throughput bidirectional traffic,” *Sensors*, vol. 18, no. 2, p. 325, Jan. 2018.

[69] Ivo B. F. de Almeida, Luciano L. Mendes, Joel J. P. C. Rodrigues, and Mauro A. A. da Cruz, “5G Waveforms for IoT Applications,” *IEEE CommunIcations Surveys & Tutorials*, vol. 21, no. 3, pp. 2554–2567, 3rd quart. 2019.

[70] L. Zhang, Y. C. Liang, and D. Niyato, “6G Visions: Mobile ultra-broadband, super internet-of-things, and artificial intelligence,” *China Communications*, vol. 16, no. 8, pp. 1–14, Aug. 2019.

[71] B. Nour, K. Sharif, F. Li, S. Yang, H. Moungla, and Y. Wang, “ICN Publisher-Subscriber Models: Challenges and Group-based Communication,” *IEEE Network*, vol. 33, no. 6, pp. 156–163, Nov.-Dec. 2019.

[72] N. Koutroumanis, G. M. Santipantakis, A. Glenis, C. Doulkeridis, and G. A. Vouros, “Scalable enrichment of mobility data with weather information,” *Geoinformatica*, Sep. 2020.

[73] Philippe Dobbelaere and Kyumars S. Esmaili, “Kafka versus RabbitMQ,” in *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*, 2017, Barcelona, Spain, 19-23 Jun. 2017, pp. 227-238.

[74] R. Balaji, A. V. R. Mayuri, N. Ramadevi, and R. Anirudh Reddy, “Advanced implementation patterns of internet of things with MQTT providers in the cutting edge communications,” *Materials Today: Proceedings*, Dec. 2020.

[75] C. V. Phung, J. Dizdarevic, and A. Jukan, “An Experimental Study of Network Coded REST HTTP in Dynamic IoT Systems,” in *2020 IEEE International Conference on Communications (ICC 2020)*, Dublin, Ireland, 7-11 Jun. 2020, pp. 1-6.

[76] V. Rampérez, J. Soriano, D. Lizcano, and J. A. Lara, “An innovative approach to improve elasticity and performance of message brokers for green smart cities,” in Proceedings of the Fourth International Conference on Engineering & MIS (ICEMIS 18), Istanbul, Tukey, 19-20 Jun. 2018, pp. 1-5.

[77] D. C. G. Valadares, M. S. L. da Silva, A. E. M. Brito, and E. M. Salvador, “Achieving Data Dissemination with Security using FIWARE and Intel Software Guard Extensions (SGX),” in *2018 IEEE Symposium on Computers and Communications (ISCC 2018)*, Natal, Brazil, 25-28 Jun. 2018, pp. 1–7.

[78] D. Glaroudis, A. Iossifides, and P. Chatzimisios, “Survey, comparison and research challenges of IoT application protocols for smart farming,” *Computer Networks*, vol. 168, p. 107037, Feb. 2020.

[79] N. Nikolov, “Research of MQTT, CoAP, HTTP and XMPP IoT Communication protocols for Embedded Systems,” in *29th International Scientific Conference Electronics*, Sozopol, Bulgaria, 16-18 Sep. 2020, pp. 1-4.

[80] C. Gomez, A. Arcia-Moret, and J. Crowcroft, “TCP in the Internet of Things: From Ostracism to Prominence,” *IEEE Internet Computing*, vol. 22, no. 1, pp. 29–41, Jan./Feb. 2018.

[81] M. Iglesias-Urkia, D. Casado-Mansilla, S. Mayer, J. Bilbao, and A. Urbieta, “Integrating Electrical Substations Within the IoT Using IEC 61850, CoAP, and CBOR,” *IEEE Internet*

of Things Journal, vol. 6, no. 5, pp. 7437–7449, Oct. 2019.

[82] M. Koster, SmartThings, A. Keranen, J. Jimenez, and Ericsson, “draft-ietf-core-coap-pubsub-09 - Publish-Subscribe Broker for the Constrained Application Protocol (CoAP),”. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-core-coap-pubsub/. [Accessed: 10-Feb-2021].

[83] S. N. Firdous, Z. Baig, C. Valli, and A. Ibrahim, “Modelling and evaluation of malicious attacks against the IoT MQTT protocol,” *2017 IEEE International Conference on Internet Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Exeter, Uk, 21-23 Jun. 2017, pp. 748-755*.

[84] “RFC 7252 - The Constrained Application Protocol (CoAP).” [Online]. Available: https://tools.ietf.org/html/rfc7252. [Accessed: 09-Mar-2021].

[85] W. U. Rahman, Y.-S. Choi, and K. Chung, “Performance Evaluation of Video Streaming Application Over CoAP in IoT,” *IEEE Access*, vol. 7, pp. 39852–39861, Mar. 2019.

[86] S. Profanter, A. Tekat, K. Dorofeev, M. Rickert, and A. Knoll, “OPC UA versus ROS, DDS, and MQTT: Performance Evaluation of Industry 4.0 Protocols,” in *2019 IEEE International Conference on Industrial Technology (ICIT)*, Melbourne, VIC, Australia, 13-15 Feb. 2019, pp. 955-962.

[87] Z. Kang, R. Canady, A. Dubey, A. Gokhale, S. Shekhar, and M. Sedlacek, “A Study of Publish/Subscribe Middleware Under Different IoT Traffic Conditions,” in *Proceedings of the International Workshop on Middleware and Applications for the Internet of Things M4IoT'20*, Delft, Netherlands, 7-11 Dec. 2020, pp. 7–12.

[88] A. Hakiri, P. Berthou, A. Gokhale, and S. Abdellatif, “Publish/subscribe-enabled software defined networking for efficient and scalable IoT communications,” *IEEE Communications Magazine*, vol. 53, no. 9, pp. 48–54, Sep. 2015.

[89] P. Murugesan, S. Chinnappa, A. Alaerjan, and D.-K. Kim, “Adopting Attribute-Based Access Control to Data Distribution Service,” in *2017 International Conference on Software Security and Assurance (ICSSA)*, Altoona, PA, USA, 24-25 Jul. 2017, pp. 112–115.

[90] R. Priyamvadaa, “Temperature and Saturation level monitoring system using MQTT for COVID-19,” in *2020 International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, Bangalore, India, 12-13 Nov. 2020, pp. 17–20.

[91] Object Management Group, “The Real-time Publish-Subscribe Protocol (RTPS) DDS Interoperability Wire Protocol Specification,”Milford, MA, USA: Object Management Group, 2014.

[92] H. Wang, D. Xiong, P. Wang, and Y. Liu, “A Lightweight XMPP Publish/Subscribe Scheme for Resource-Constrained IoT Devices,” *IEEE Access*, vol. 5, pp. 16393–16405, Aug. 2017.

[93] “RFC 6120 - Extensible Messaging and Presence Protocol (XMPP): Core.” [Online]. Available: https://tools.ietf.org/html/rfc6120. [Accessed: 09-Mar-2021].

[94] “RFC 6121 - Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence.” [Online]. Available: https://tools.ietf.org/html/rfc6121. [Accessed: 09-Mar-2021].

[95] P. Anantharaman, M. Locasto, G. F. Ciocarlie, and U. Lindqvist, “Building hardened internet-of-things clients with language-theoretic security,” *2017 IEEE Security and Privacy Workshops (SPW)*, San Jose, CA, USA, 25-25 May 2017, pp. 120–126.

[96] M. El Ouadghiri, B. Aghoutane, and N. El Farissi, “Communication model in the internet of things,” *Procedia Computer Science*, vol. 177, pp. 72–77, 2020.

[97] “RFC 6455 - The WebSocket Protocol.” [Online]. Available: https://tools.ietf.org/html/rfc6455. [Accessed: 09-Mar-2021].

[98] A. S. Abdelfattah, T. Abdelkader, and E.-S. M. EI-Horbaty, “RAMWS: Reliable approach using middleware and WebSockets in mobile cloud computing,” *Ain Shams Engineering Journal*, vol. 11, no. 4, pp. 1083–1092, Dec. 2020.

[99] V. Pimentel and B. G. Nickerson, “Communicating and displaying real-time data with WebSocket,” *IEEE Internet Computing*, vol. 16, no. 4, pp. 45–53, Jul. 2012.

[100] Y. Chen and T. Kunz, “Performance evaluation of IoT protocols under a constrained wireless access network,” *2016 International Conference on Selected Topics in Mobile & Wireless Networking (MoWNeT)*, Cairo, Egypt, 11-13 Apr. 216, pp. 1-7.

[101] S. Mijovic, E. Shehu, and C. Buratti, “Comparing application layer protocols for the Internet of Things via experimentation,” *2016 IEEE 2nd International Forum on Research Technologies for Society and Industry Leveraging a Better Tomorrow (RTSI)*, Bologna, Italy, 7-9 Sep. 2016, pp. 1-5.

[102] C. Kalmanek, “A retrospective view of ATM,” ACM SIGCOMM *Comput. Communication Review*, vol. 32, no. 5, pp. 13–19, Nov. 2002.

[103] B. Mishra and A. Kertesz, “The Use of MQTT in M2M and IoT Systems: A Survey,” *IEEE Access*, vol. 8, pp. 201071–201086, 2020.

[104] G. Kim, S. Kang, J. Park, and K. Chung, “An MQTT-Based Context-Aware Autonomous System in oneM2M Architecture,” *IEEE Internet Things Journal*, vol. 6, no. 5, pp. 8519–8528, Oct. 2019.

[105] M. Yannuzzi, R. Irons-Mclean, F. V. Lingen, S. Raghav, A. Somaraju, C. Byers, T. Zhang, A. Jain, J. Curado, D. Carrera, O. Trullols, and S. Alonso, “Toward a converged OpenFog and ETSI MANO architecture,” in *2017 IEEE Fog World Congress (FWC)*, Santa Clara, CA, USA, 30 Oct. - 1 Nov. 2017, pp. 1–6.

[106] O. Tomanek and L. Kencl, “Security and privacy of using AllJoyn IoT framework at home and beyond,” *2016 International Conference on Intelligent Green Building and Smart Grid (IGBSG)*, Prague, Czech Republic, 27-29 Jun. 2016, pp. 1-6.

[107] J.-C. Lee, J.-H. Jeon, and S.-H. Kim, “Design and implementation of healthcare resource model on IoTivity platform,” in *2016 International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju, South Korea, 18-21 Oct. 2016, pp. 887–891.

[108] V. P. Singh, V. T. Dwarakanath, P. Haribabu, and N. S. C. Babu, “IoT standardization

efforts — An analysis," in *2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)*, Bengaluru, India, 17-19 Aug. 2017, pp. 1083–1088.

[109] OMA SpecWorks, "IPSO Alliance Merges with Open Mobile Alliance to Form OMA SpecWorks.", [Online]. Available: https://omaspecworks.org/ipso-alliance-merges-with-open-mobile-alliance-to-form-oma-specworks. [Accessed: 09-Mar-2021].

[110] S. Park, "OCF: A New Open IoT Consortium," in *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, Taipei, Taiwan, 27-29 Mar. 2017, pp. 356–359.

[111] E. C. do Rosario, V. H. D. Davila, T. B. da Silva, and A. M. Alberti, "A Docker-Based Platform for Future Internet Experimentation: Testing NovaGenesis Name Resolution," in *2019 IEEE Latin-American Conference on Communications (LATINCOM)*, Salvador, Brazil, 11-13 Nov. 2019, , pp. 1–5.

[112] Antonio M. Alberti, Gabriel D. Scarpioni, Vaner J. Magalhaes, Arismar S. Cerqueira, Joel J. P. C. Rodrigues, and Rodrigo Da Rosa Righi, "Advancing NovaGenesis Architecture Towards Future Internet of Things," *IEEE Internet Things Journal*, vol. 6, no. 1, pp. 215–229, Feb. 2019.

[113] A. Sagheer, M. Mohammed, K. Riad, and M. Alhajhoj, "A Cloud-Based IoT Platform for Precision Control of Soilless Greenhouse Cultivation," *Sensors*, vol. 21, no. 1, p. 223, Dec. 2020.

[114] P. Agarwal and M. Alam, "Investigating IoT Middleware Platforms for Smart Application Development," in *Lecture Notes Civil Engineering*, Springer Singapore, 2020, pp. 231–244.

[115] C. Vielma, A. Verma, and D. Bein, "Single and Multibranch CNN-Bidirectional LSTM for IMDb Sentiment Analysis," in *Advances in Intelligent Systems and Computing*, Springer International Publishing, 2020, pp. 401–406.

[116] L. T. De Paolis, V. De Luca, and R. Paiano, "Sensor data collection and analytics with thingsboard and spark streaming," in *2018 IEEE Workshop on Environmental, Energy, and Structural Monitoring Systems (EESMS)*, Salerno, Italy, 21-22 Jun. 2018, pp. 1–6.

[117] A. Mynzhasova*, C. Radojicic, C. Heinz, J. Kölsch, C. Grimm, J. Rico, K. Dickerson, R. García-Castro, and V. Oravec*, "Drivers, standards and platforms for the IoT: Towards a digital VICINITY," in *2017 Intelligent Systems Conference (IntelliSys)*, London, UK, 7-8 Sep. 2017, pp. 170–176.

[118] L. Enciso and A. Vargas, "Interface with Ubidots for a fire alarm system using WiFi," in *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*, Caceres, Spain, 13-16 Jun. 2018, pp. 1–6.

[119] N. Sinha, K. E. Pujitha, and J. S. R. Alex, "Xively based sensing and monitoring system for IoT," in *2015 International Conference on Computer Communication and Informatics (ICCCI)*, Coimbatore, India, 8-10 Jan. 2015, pp. 1–6.

[120] P. Fernández, J. M. Santana, S. Ortega, A. Trujilo, J. P. Suárez, C. Domínguez, J. Santana, and A. Sánchez, "Smartport: A platform for sensor data monitoring in a seaport based on FIWARE," *Sensors*, vol. 16, no. 3, p. 417, Mar. 2016.

[121] F. H. Cabrini, A. De Barros Castro Filho, F. V. Filho, S. T. Kofuji, and A. R. L. P. Moura, “Helix SandBox: An open platform to fast prototype smart environments applications,” in *2019 IEEE 1st Sustainable Cities Latin America Conference (SCLA)*, Arepira, Peru, 26-19 Aug. 2019, pp. 1–6, 2019.

[122] C. Kamienski, J.-P. Soininen, M. Taumberger, R. Dantas, A. Toscano, T. S. Cinotti, R. F. Maia, and A. T. Neto , “Smart water management platform: IoT-based precision irrigation for agriculture,” *Sensors*, vol. 19, no. 2, p. 276, Jan. 2019.

[123] Fiware, “Fiware-Orion.” [Online]. Available: https://fiware-orion.readthedocs.io/en/develop/. [Accessed: 14-Mar-2021].

[124] Fiware, “Fiware-STH-Comet.” [Online]. Available: https://fiware-sth-comet.readthedocs.io/en/latest/. [Accessed: 14-Mar-2021].

[125] K. Gunasekera, A. N. Borrero, F. Vasuian, and K. P. Bryceson, “Experiences in building an IoT infrastructure for agriculture education,” *Procedia Computer Science*, vol. 135, pp. 155–162, 2018.

[126] A. Pereira-Vale, G. Marquez, H. Astudillo, and E. B. Fernandez, “Security Mechanisms Used in Microservices-Based Systems: A Systematic Mapping,” in *2019 XLV Latin American Computing Conference (CLEI)*, Panama, Panama, 30 Sep. - 4 Oct. 2019, pp. 1–10.

[127] SiteWhere, “SiteWhere | The Open Platform for the Internet of Things.” [Online]. Available: http://www.sitewhere.org/. [Accessed: 16-Mar-2021].

[128] M. Eisenhauer, P. Rosengren, and P. Antolin, “A Development Platform for Integrating Wireless Devices and Sensors into Ambient Intelligence Systems,” in *2009 6th IEEE Annual Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops*, Rome, Italy, 22-26 June 2009, pp. 1–3.

[129] A. C. Tsolakis, I. Moschos, A. Zerzelidis, P. Tropios, S. Zikos, A. Tryferidis, S. Krinidis, D. Ionnidis, and D. Tzovaras, “Occupancy-based decision support system for building management: From automation to end-user persuasion,” *International Journal of Energy Research*, vol. 43, no. 6, pp. 2261–2280, Mar. 2019.

[130] D. Raca, M. Manifacier, and J. J. Quinlan, “goDASH — GO Accelerated HAS Framework for Rapid Prototyping,” in *2020 Twelfth International Conference on Quality of Multimedia Experience (QoMEX)*, Athlone, Ireland, 26-28 May 2020, pp. 1–4.

[131] J. D. Leon, *Security with Go: Explore the power of Golang to secure host, we, and cloud services*. Birmingham - UK: Packt Publishing, Jan. 2018.

[132] Linksmart, “LinkSmart® Documentation Home - Home - LinkSmart® Docs.” [Online]. Available: https://docs.linksmart.eu/. [Accessed: 22-Feb-2021].

[133] Konker Labs, “Konker - Build your IoT solutions in days instead of months.” [Online]. Available: http://www.konkerlabs.com/. [Accessed: 22-Feb-2021].

[134] C. Gyorodi, R. Gyorodi, G. Pecherle, and A. Olah, “A comparative study: MongoDB vs. MySQL,” in *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*, Oradea, Romania, 11-12 Jun. 2015, pp. 1–6.

[135] S. H. Aboutorabi, M. Rezapour, M. Moradi, and N. Ghadiri, “Performance evaluation

of SQL and MongoDB databases for big e-commerce data," in *2015 International Symposium on Computer Science and Software Engineering (CSSE)*, Tabriz, Iran, 18-19 Aug. 2015, pp. 1–7.

[136] O. Ethelbert, F. F. Moghaddam, P. Wieder, and R. Yahyapour, "A JSON Token-Based Authentication and Access Management Schema for Cloud SaaS Applications," in *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, Prague, Czech Republic, 21-23 Aug. 2017, pp. 47–53.

[137] S. Cirani, M. Picone, P. Gonizzi, L. Veltri, and G. Ferrari, "IoT-OAS: An oauth-based authorization service architecture for secure services in IoT scenarios," *IEEE Sensors Journal*, vol. 15, no. 2, pp. 1224–1234, Feb. 2015.

[138] C. Pereira, J. Cardoso, A. Aguiar, and R. Morla, "Benchmarking Pub/Sub IoT middleware platforms for smart services," *Journal of Reliable Intelligent Environments*, vol. 4, no. 1, pp. 25–37, Feb. 2018.

[139] C. Akasiadis, V. Pitsilis, and C. D. Spyropoulos, "A multi-protocol IoT platform based on open-source frameworks," *Sensors*, vol. 19, no. 19, p. 4217, Sep. 2019.

[140] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347–2376, 4th Quart. 2015.

[141] A. H. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and M. Z. Sheng, "IoT Middleware: A Survey on Issues and Enabling technologies," *IEEE Internet Things Journal*, vol. 4, no. 1, pp. 1–20, Feb. 2017.

[142] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Cla, "Middleware for internet of things: A survey," *IEEE Internet of Things Journal*, vol. 3, no. 1, pp. 70–95, Feb. 2016.

[143] G. Fersi, "Middleware for Internet of Things: A Study," in *2015 International Conference on Distributed Computing in Sensor Systems*, Fortaleza, Brazil, 10-12 Jun. 2015, pp. 230–235.

[144] A. A. Ismail, H. S. Hamza, and A. M. Kotb, "Performance Evaluation of Open Source IoT Platforms," in *2018 IEEE Global Conference on Internet of Things (GCIoT)*, Alexandria, Egypt, 5-7 Dec. 2018, pp. 1–5.

[145] J. P. Brans and P. Vincke, "A Preference Ranking Organisation Method: (The PROMETHEE Method for Multiple Criteria Decision-Making)," *Management Science*, vol. 31, no. 6, pp. 647–656, Jun. 1985.

[146] M. Behzadian, R. B. Kazemzadeh, A. Albadvi, and M. Aghdasi, "PROMETHEE: A comprehensive literature review on methodologies and applications," *European Journal of Operational Research*, vol. 200, no. 1, pp. 198–215, Jan. 2010.

[147] R. Nawaratne, D. Alahakoon, D. De Silva, P. Chhetri, and N. Chilamkurti, "Self-evolving intelligent algorithms for facilitating data interoperability in IoT environments," *Future Generation Computer Systems*, vol. 86, pp. 421–432, Sep. 2018.

[148] B. Farahani, F. Firouzi, V. Chang, M. Badaroglu, N. Constant, and K. Mankodiya, "Towards fog-driven IoT eHealth: Promises and challenges of IoT in medicine and healthcare," *Future Generation Computer Systems*, vol. 78, part 2, pp. 659–676, Jan.

2018.

[149] I. Unwala, Z. Taqvi, and J. Lu, “Thread: An IoT protocol,” in 2018 *IEEE Green Technologies Conference (Greentech)*, Austin, TX, USA, 4-6 Apr. 2018, pp. 161-167.

[150] G. Aloi, G. Fortino, R. Gravina, P. Pace, W. Russo, and C. Savaglio, “Enabling IoT interoperability through opportunistic smartphone-based mobile gateways,” *Journal of Network and Computer Applications*, vol. 81, October, pp. 74–84, pp. 74-84, Mar. 2017.

[151] T. Egyedi and S. Muto, “Standards for ICT - A green strategy in a grey sector,” in *2011 7th International Conference on Standardization and Innovation in Information Technology (SIIT)*, 2011, Berlin, Germany, 28-30 Sep. 2011, pp. 1–10.

[152] A. Farahzadi, P. Shams, J. Rezazadeh, and R. Farahbakhsh, “Middleware technologies for cloud of things-a survey,” *Digital Communications and Networks*, vol. 4, no. 3, pp. 176–188, Aug. 2018.

[153] A. Al-Fuqaha, A. Khreishah, M. Guizani, A. Rayes, and M. Mohammadi, “Toward better horizontal integration among IoT services,” *IEEE Communications Magazine*, vol. 53, no. 9, pp. 72–79, Sep. 2015.

[154] N. Naik, “Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP,” In *2017 IEEE International Symposium on Systems Engineering (ISSE)*, Vienna, Austria 11-13 Oct. 2017, pp. 1–7.

[155] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, “A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration,” *ACM Computing Surveys*, vol. 51, no. 6, pp. 1–29, Feb. 2019.

[156] Ejaz Ahmed, Ibrar Yaqoob, Ibrahim A. T. Hashem, Imran Khan, Abdelmuttlib I. A. Ahmed, Muhammad Imran, Athanasios V. Vasilakos, “The role of big data analytics in Internet of Things,” *Computer Networks*, vol. 129, Part 2, pp. 459–471, Dec. 2017.

[157] V. Issarny, A. Bennaceur, and Y.-D. Bromberg, “Middleware-Layer Connector Synthesis: Beyond State of the Art in Middleware Interoperability,” in *Formal Methods for Eternal Networked Software Systems*, Springer Berlin Heidelberg, 2011, pp. 217–255.

[158] Eclipse Foundation, “Ponte - Bringing Things to REST developers.” [Online]. Available: https://www.eclipse.org/ponte/. [Accessed: 16-Mar-2021].

[159] W. L. D. a M. Macêdo, T. Rocha, and E. D. Moreno, “GoThings An Application-layer Gateway Architecture for the Internet of Things,” in *11th International Conference on Web Information Systems and Technologies (Webist2015)*, Lisbon, Portugal, 2015, pp. 135–140.

[160] M. Collina, G. E. Corazza, and A. Vanelli-Coralli, “Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST,” in *2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC)*, Sydney, NSW, Australia, 9-12 Sep. 2017, pp. 36–41.

[161] A. La Marra, F. Martinelli, P. Mori, A. Rizos, and A. Saracino, “Improving MQTT by Inclusion of Usage Control,” in *2017 Security, Privacy, and Anonymity in Computation Communication, and Storage (SpaCCS)*, Guangzhou, China, 12-15 Dec. 2017, pp. 545–

560.

[162] B. Al-Madani and H. Ali, “Data Distribution Service (DDS) based implementation of Smart grid devices using ANSI C12.19 standard,” *Procedia Computer Science*, vol. 110, pp. 394–401, 2017.

[163] M. Iglesias-Urkia, A. Orive, A. Urbieta, and D. Casado-Mansilla, “Analysis of CoAP implementations for industrial Internet of Things: a survey,” *Procedia Computer Science*, vol. 109, pp. 188–195, 2017.

[164] “TooTallNate/Java-WebSocket: A barebones WebSocket client and server implementation written in 100% Java.” [Online]. Available: https://github.com/TooTallNate/Java-WebSocket. [Accessed: 20-Mar-2021].

[165] “GitHub - vivirodrigues/MiddleBridge: This project does the conversion of messages sent by an IoT protocol to HTTP messages.” [Online]. Available: https://github.com/vivirodrigues/MiddleBridge. [Accessed: 20-Mar-2021].

[166] T. Zahariadis, A. Papadakis, F. Alvarez, J. Gonzalez, F. Lopez, F. Facca, Y. Al-Hazmi, “FIWARE lab: Managing resources and services in a cloud federation supporting future internet applications,” in 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, London, UK, 8-11 Dec. 2014, pp. 792–799.

[167] “emqtt/mqtt-jmeter: MQTT JMeter Plugin.” [Online]. Available: https://github.com/emqtt/mqtt-jmeter. [Accessed: 20-Mar-2021].

[168] “pjtr / jmeter-websocket-samplers — Bitbucket.” [Online]. Available: https://bitbucket.org/pjtr/jmeter-websocket-samplers/src/master/. [Accessed: 20-Mar-2021].

[169] “Tanganelli/jmeter-coap: jmeter plugin to evaluate CoAP servers.” [Online]. Available: https://github.com/Tanganelli/jmeter-coap. [Accessed: 20-Mar-2021].

[170] D. E. Kouicem, A. Bouabdallah, and H. Lakhlef, “Internet of things security: A top-down survey,” *Comput. Networks*, vol. 141, pp. 199–221, Aug. 2018.

[171] M. binti Mohamad Noor and W. H. Hassan, “Current research on Internet of Things (IoT) security: A survey,” *Computer Networks*, vol. 148, pp. 283–294, Jan. 2019.

[172] OpenJS Foundation, “About | Node.js.” [Online]. Available: https://nodejs.org/en/about/. [Accessed: 20-Mar-2021].

[173] V. Araujo, K. Mitra, S. Saguna, and C. Åhlund, “Performance evaluation of FIWARE: A cloud-based IoT platform for smart cities,” *Journal of Parallel Distributed Computing*, vol. 132, pp. 250–261, Oct. 2019.

[174] S. Nanz and C. A. Furia, “A Comparative Study of Programming Languages in Rosetta Code,” in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 2015, Florence, Italy, 16-24 May 2015, pp. 778–788.

[175] C. Thirumalai, P. A. Reddy, and Y. J. Kishore, “Evaluating software metrics of gaming applications using code counter tool for C and C++ (CCCC),” in *2017 International conference of Electronics, Communication and Aerospace Technology (ICECA)*, Coimbatore, India, 20-22 Apr. 2017, pp. 180–184.

[176] T. Sharma, M. Fragkoulis, and D. Spinellis, “House of Cards: Code Smells in Open-Source C# Repositories,” in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Toronto, ON, Canada, 9-10 Nov. 2017, pp. 424–429.

[177] D. Beyer, “Automatic Verification of C and Java Programs: SV-COMP 2019,” in 2019 *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Prague, Czech Republic, 6-11 Apr. 2019, pp. 133–155.

[178] D. Syme, “The early history of F#,” *Proceedings of the ACM Programing on Languages*, vol. 4, no. HOPL, pp. 1–58, Jun. 2020.

[179] S. Weirich, P. Choudhury, A. Voizard, and R. A. Eisenberg, “A role for dependent types in Haskell,” *Proceedings of the ACM Programing on Languages*, vol. 3, no. ICFP, pp. 1–29, Jul. 2019.

[180] Q. Sun, T. C. Berkelbach, N. S. Blunt, G. H. Booth, S. Guo, Z. Li, J. Liu, J. D. McClain, E. R. Sayfutyarova, S. Sharma, S. Wouters, and G. K. Chan, “PYSCF: the Python-based simulations of chemistry framework,” *WIREs Computational Molecular Science*, vol. 8, no. 1, p. e1340, Sep. 2018, doi: 10.1002/wcms.1340.

[181] J. Hudgens, *Comprehensive Ruby Programming*. Birmingham - UK: Packt Publishing, Jun. 2017.

[182] “Rosetta Code.” [Online]. Available: http://rosettacode.org/wiki/Rosetta_Code. [Accessed: 08-Mar-2021].

[183] L. Prechelt, “An empirical comparison of seven programming languages,” *Computer*, vol. 33, no. 10, pp. 23–29, Oct. 2000.

[184] C. Nguyen-Thanh, V. P. Nguyen, A. de Vaucorbeil, T. Kanti Mandal, and J.-Y. Wu, “Jive: An open source, research-oriented C++ library for solving partial differential equations,” *Advances in Engineering Software*, vol. 150, p. 102925, Dec. 2020.

[185] S. Suwazono and H. Arao, “A newly developed free software tool set for averaging electroencephalogram implemented in the Perl programming language,” *Heliyon*, vol. 6, no. 11, p. e05580, Nov. 2020.

[186] E. K. Gerger, “Business programming with REXX: Bringing programming to business students,” *2017 8th International Conference on Information, Intelligent Systems & Applications (IISA)*, Larnaca, Cyprus, 27-30 Aug. 2017, pp. 1–5.

[187] V. Ivanova, A. Boneva, Y. Doshev, S. Ivanov, and P. Vasilev, “Multifunctional Operating Station Based on Tcl/Tk and Its Applications,” in *2019 Big Data, Knowledge and Control Systems Engineering (BdKCSE)*, Sofia, Bulgaria, 21-22 Nov. 2019, pp. 1–7.

[188] S. B. Aruoba and J. Fernández-Villaverde, “A comparison of programming languages in macroeconomics,” *Journal of Economic Dynamics and Control*, vol. 58, pp. 265–273, Sep. 2015.

[189] L. Naterop, A. Signer, and Y. Ulrich, “handyG—Rapid numerical evaluation of generalised polylogarithms in Fortran,” *Computer Physics Communications*, vol. 253, p. 107165, Aug. 2020.

[190] T. Besard, C. Foket, and B. De Sutter, “Effective Extensible Programming: Unleashing

Julia on GPUs," *IEEE Transactions on Parallel Distributed Systems*, vol. 30, no. 4, pp. 827–841, Apr. 2019.

[191] A. R. Jalalvand, M. Roushani, H. C. Goicoechea, D. N. Rutledge, and H. W. Gu, "MATLAB in electrochemistry: A review," *Talanta*, vol. 194, pp. 205–225, Mar. 2019.

[192] A. K. Cyrol, M. Mitter, and N. Strodthoff, "FormTracer. A mathematica tracing package using FORM," *Computer Physics Communications*, vol. 219, pp. 346–352, Oct. 2017.

[193] C. Chen, T. R. Razak, and J. M. Garibaldi, "FuzzyR: An Extended Fuzzy Logic Toolbox for the R Programming Language," in *2020 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, Glasgow, UK, 19-24 Jul. 2020, pp. 1–8.

[194] J. J. Merelo, P. Castillo, I. Blancas, G. Romero, P. García-Sanchez, A. Fernández-Ares, V. Rivas, and M. Gárcia-Valdez, "Benchmarking Languages for Evolutionary Algorithms," in 2016 *EvoApplications: European Conference on the Applications of Evolutionary Computation*, Porto, Portugal, 30 Mar. - 1 Apr. 2016, pp. 27–41.

[195] F. Devin, "SCALA – Functional Programming," in *TORUS 1 – Toward an Open Resource Using Services*, Wiley, pp. 207–228, Apr. 2020.

[196] H. M. Gualandi and R. Ierusalimschy, "Pallene: A companion language for Lua," *Science of Computer Programing*, vol. 189, p. 102393, Apr. 2020.

[197] R. S. Malik, J. Patra, and M. Pradel, "NL2Type: Inferring JavaScript Function Types from Natural Language Information," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, Montreal, QC, Canada, 25-31 May 2019, pp. 304–315.

[198] R. Aley, *Pro Functional PHP Programming*. Apress, 2017.

[199] K. Lei, Y. Ma, and Z. Tan, "Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js," in *2014 IEEE 17th International Conference on Computational Science and Engineering*, Chengdu, China, 19-21 Dec. 2014, pp. 661–668.

[200] Jie Wang, Wensheng Dou1, Yu Gao, Chushu Gao, Feng Qin, Kang Yin, and Jun Wei, "A comprehensive study on real world concurrency bugs in Node.js," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Urbana, IL, USA, 30 Oct. - 3 Nov. 2017, pp. 520–531.

[201] Y. A. Auliya, Y. Nurdinsyah, and D. A. R. Wulandari, "Performance Comparison of Docker and LXD with ApacheBench," *Journal of Physics: Conference Series*, vol. 1211, p. 012042, Apr. 2019.

[202] G. Marceau, "Guillaume Marceau: The speed, size and dependability of programming languages." [Online]. Available: http://blog.gmarceau.qc.ca/2009/05/speed-size-and-dependability-of.html. [Accessed: 13-Jan-2021].

[203] TechEmpower, "Round 17 results - TechEmpower Framework Benchmarks." [Online]. Available: https://www.techempower.com/benchmarks/#section=data-r17&hw=ph&test=query. [Accessed: 13-Jan-2021].

[204] StackOverflow, "Stack Overflow Developer Survey 2019." [Online]. Available: https://insights.stackoverflow.com/survey/2019. [Accessed: 13-Jan-2021].

[205] A. Wajid, P. Junjun, A. Akbar, and M. A. Mughal, “WebGraveStone: An online gravestone design system based on jQuery and MVC framework,” in *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, Sukkur, Pakistan, 3-4 Mar. 2018, pp. 1–6.

[206] A. Banks and E. Porcello, *Learning React: Functional Web Development with React and Redux 1st Edition*. O’Reilly Media, 2017.

[207] M. Ramos, M. T. Valente, and R. Terra, “AngularJS Performance: A Survey Study,” *IEEE Software*, vol. 35, no. 2, pp. 72–79, Mar./Apr. 2018.

[208] B. Nelson, *Getting to Know Vue.js*. Apress, 2018.

[209] X. Yu and Q. Zhou, “Design and Implementation of Supply Chain Management System Based on ASP.NET,” in *2020 International Wireless Communications and Mobile Computing (IWCMC)*, Limassol, Cyprus, 15-19 Jun. 2020, pp. 1454–1457.

[210] E. H. Hahn, *Express in Action: Writing, building, and testing Node.js applications*. Manning Publications Co. 2016.

[211] M. Gajewski and W. Zabierowski, “Analysis and Comparison of the Spring Framework and Play Framework Performance, Used to Create Web Applications in Java,” in *2019 IEEE XVth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, Polyana, Ukraine, 22-26 May 2019, pp. 170–173.

[212] R. Correia and E. Adachi, “Detecting Design Violations in Django-based Web Applications,” in *Proceedings of the XIII Brazilian Symposium on Software Components, Architectures, and Reuse - SBCARS ’19*, Salvador, Brazil, 19-23 Oct. 2019, pp. 33–42.

[213] M. R. Mufid, A. Basofi, M. U. H. Al Rasyid, I. F. Rochimansyah, and A. Rokhim, “Design an MVC Model using Python for Flask Framework Development,” in *2019 International Electronics Symposium (IES)*, Surabaya, Indonesia, 27-28 Sep. 2019, pp. 214–219.

[214] A. Sunardi and Suharjito, “MVC Architecture: A Comparative Study Between Laravel Framework and Slim Framework in Freelancer Project Monitoring System Web Based,” *Procedia Computer Science*, vol. 157, pp. 134–141, 2019.

[215] P. Łuczak, A. Poniszewska-Maranda, and V. Karovič, “The Process of Creating Web Applications in Ruby on Rails,” in Developments in Information & Knowledge Management for Business Applications, Springer International Publishing, 2020, pp. 371–401.

[216] L. Abbade, “IoT Middleware Performanc.” [Online]. Available: https://github.com/LRAbbade/iot_middleware_ performance. [Accessed: 13-Jan-2021].

[217] M. Jones, “RFC 7519 - JSON Web Token (JWT).” [Online]. Available: https://tools.ietf.org/html/rfc7519%0D. [Accessed: 13-Jan-2021].

[218] S. H. Aboutorabi, M. Rezapour, M. Moradi, and N. Ghadiri, “Performance evaluation of SQL and MongoDB databases for big e-commerce data,” in *2015 International Symposium on Computer Science and Software Engineering (CSSE)*, Tabriz, Iran, 18-19 Aug. 2015, pp. 1–7.

[219] Y. Li and S. Manoharan, “A performance comparison of SQL and NoSQL databases,” in *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, Victoria, BC, Canada, 27-29 Aug. 2013, pp. 15–19.

[220] C. Gyorodi, R. Gyorodi, G. Pecherle, and A. Olah, “A comparative study: MongoDB vs. MySQL,” in *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*, Oradea, Romania, 11-12 Jun. 2015, pp. 1–6.

[221] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An updated performance comparison of virtual machines and Linux containers,” in *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Philadelphia, PA, USA, 29-31 Mar. 2015, pp. 171–172.

[222] Locust, “Locust - A modern load testing framework.” [Online]. Available: https://locust.io/. [Accessed: 13-Jan-2021].

[223] Kubernetes, “Kubernetes.” [Online]. Available: https://kubernetes.io/. [Accessed: 13-Jan-2021].

[224] L. Abbade, “GitHub - LRAbbade/load_testing: Load testing with Locust and Kubernetes test.” [Online]. Available: https://github.com/LRAbbade/load_testing. [Accessed: 13-Jan-2021].

[225] D. Taibi, V. Lenarduzzi, and C. Pahl, “Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation,” *IEEE Cloud Compuing*, vol. 4, no. 5, pp. 22–32, Sep./Oct. 2017.

[226] A. Balalaie, A. Heydarnoori, and P. Jamshidi, “Microservices Architecture Enables DevOps: An Experience Report on Migration to a Cloud-Native Architecture,” *IEEE Software*, vol. 33, no. 3, pp. 42–52, May-Jun. 2016.

[227] O. Zimmermann, “Microservices tenets,” *Computer Science Research and Development*, vol. 32, no. 3–4, pp. 301–310, Nov. 2016.

[228] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, L. Safina, “Microservices: Yesterday, Today, and Tomorrow,” in *Present and Ulterior Software Engineering*, Springer International Publishing, 2017, pp. 195–216.

[229] P. Di Francesco, P. Lago, and I. Malavolta, “Architecting with microservices: A systematic mapping study,” *Journal of Systems and Software*, vol. 150. pp. 77–97, Apr. 2019.

[230] M. S. Chae, H. M. Lee, and K. Lee, “A performance comparison of linux containers and virtual machines using Docker and KVM,” *Cluster Computing*, vol. 22, no. s1, pp. 1765–1775, Dec. 2017.

[231] S. Kwon and J. H. Lee, “DIVDS: Docker Image Vulnerability Diagnostic System,” *IEEE Access*, vol. 8, pp. 42666–42673, 2020.

[232] A. M. Potdar, N. D G, S. Kengond, and M. M. Mulla, “Performance Evaluation of Docker Container and Virtual Machine,” *Procedia Computer Science*, vol. 171, pp. 1419–1428, 2020.

[233] V. Medel, R. Tolosana-Calasanz, J. Á. Bañares, U. Arronategui, and O. F. Rana,

"Characterising resource management performance in Kubernetes," *Computers and Electrical Engineering*, vol. 68. pp. 286–297, May 2018.

[234] S. Taherizadeh and M. Grobelnik, "Key influencing factors of the Kubernetes auto-scaler for computing-intensive microservice-native cloud-based applications," *Advances in Engineering Software*, vol. 140, p. 102734, Feb. 2020.

[235] D. A. Pereira, W. Ourique de Morais, and E. Pignaton de Freitas, "NoSQL real-time database performance comparison," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 33, no. 2, pp. 144–156, Mar. 2017.

[236] M. T. Gonzalez-Aparicio, M. Younas, J. Tuya, and R. Casado, "Evaluation of ACE properties of traditional SQL and NoSQL big data systems," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, Limassol, Cyprus, 8-12 Apr. 2019, pp. 1988–1995.

[237] Y. Li and S. Manoharan, "A performance comparison of SQL and NoSQL databases," in 2013 *IEEE Pacific RIM Conference on Communications, Computers, and Signal Processing (PACRIM)*, Victoria, BC, Canada, 27-29 Aug. 2013, pp. 15–19.

[238] M. A. Qader, S. Cheng, and V. Hristidis, "A Comparative Study of Secondary Indexing Techniques in LSM-based NoSQL Databases," in *Proceedings of the 2018 International Conference on Management of Data*, Houston, TX, USA, 10-15 Jun. 2018, pp. 551–566.

[239] K. Sahatqija, J. Ajdari, X. Zenuni, B. Raufi, and F. Ismaili, "Comparison between relational and NOSQL databases," in *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, Croatia, 21-25 May 2018, pp. 0216–0221.

[240] A. Bandeira, C. A. Medeiros, M. Paixao, and P. H. Maia, "We need to talk about microservices: An analysis from the discussions on stackoverflow," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, Montreal, QC, Canada, 25-31 May 2019, pp. 255–259.

[241] F. Pina, J. Correia, R. Filipe, F. Araujo, and J. Cardroom, "Nonintrusive monitoring of microservice-based systems," in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, Cambridge, MA, USA, 1-3 Nov. 2018, pp. 1-8.

[242] A. Banijamali, P. Jamshidi, P. Kuvaja, and M. Oivo, "Kuksa: A Cloud-Native Architecture for Enabling Continuous Delivery in the Automotive Domain," in Product-Focused Software Process Improvement, Springer International Publishing, 2019, pp. 455–47.

[243] J. Carnell, *Spring Microservices in Action*. Shelter Island, NY, USA: Manning Publications, 2017.

[244] F. Montesi and J. Weber, "Circuit Breakers, Discovery, and API Gateways in Microservices," *arXiv Prepr. arXiv1609.05830*, Sep. 2016.

[245] A. Selva, "GitHub - moquette-io/moquette: Java MQTT lightweight broker." [Online]. Available: https://github.com/moquette-io/moquette. [Accessed: 07-Mar-2021].

[246] "In-IoT / moquette-in.iot — Bitbucket." [Online]. Available: https://bitbucket.org/In-IoT/moquette-in.iot/src/master/. [Accessed: 21-Feb-2021].

[247] B. Ashworth, "GitHub - brendanashworth/Raw-TCP-Proxy: Protocol independent TCP

proxy; based in Java.” [Online]. Available: https://github.com/brendanashworth/Raw-TCP-Proxy. [Accessed: 07-Mar-2021].

[248] “In-IoT / raw-tcp-proxy-in.iot — Bitbucket.” [Online]. Available: https://bitbucket.org/In-IoT/raw-tcp-proxy-in.iot/src/master/. [Accessed: 21-Feb-2021].

[249] Y. Wang and G. Wei, “An Implementation of CoAP-Based Resource Directory in Californium,” in *Proceedings of the 2018 2nd International Conference on Big Data and Internet of Things - BDIOT 2018*, Beijing, China, 24-27 Oct. 2018, pp. 148–152.

[250] Agorshkov23, “UDP Proxy.” [Online]. Available: https://github.com/agorshkov23/udp-duplicator. [Accessed: 07-Mar-2021].

[251] “In-IoT / californium-in.iot — Bitbucket.” [Online]. Available: https://bitbucket.org/In-IoT/californium-in.iot/src/master/. [Accessed: 21-Feb-2021].

[252] “In-IoT / udp-duplicator-in.iot — Bitbucket.” [Online]. Available: https://bitbucket.org/In-IoT/udp-duplicator-in.iot/src/master/. [Accessed: 21-Feb-2021].

[253] S. H. Aboutorabi, M. Rezapour, M. Moradi, and N. Ghadiri, “Performance evaluation of SQL and MongoDB databases for big e-commerce data,” in *2015 International Symposium on Computer Science and Software Engineering (CSSE)*, Trabiz, Iran, 18-19 Aug 2015, pp. 1-7.

[254] S. Misra and N. Saha, “Detour: Dynamic Task Offloading in Software-Defined Fog for IoT Applications,” *IEEE Journal on Selected Areas Communications*, vol. 37, no. 5, pp. 1159–1166, May 2019.

[255] S. Misra and S. Sarkar, “Priority-Based Time-Slot Allocation in Wireless Body Area Networks During Medical Emergency Situations: An Evolutionary Game-Theoretic Perspective,” *IEEE J. Biomed. Heal. Informatics*, vol. 19, no. 2, pp. 541–548, Mar. 2015, doi: 10.1109/JBHI.2014.2313374.

[256] Danielly B. Avancini, Joel J. P. C. Rodrigues, Ricardo A. L. Rabêlo, Ashok K. Das, Sergey Kozlov, and Petar Solic, “A new IoT-based smart energy meter for smart grids,” *International Journal of Energy Research*, vol. 45, no. 1, pp. 189–202, 2021, doi: 10.1002/er.5177.

[257] João B. A. Gomes, Joel J. P. C. Rodrigues, Ricardo A. L. Rabêlo, S. Tanwar, J. Al-Muhtadi, and S. Kozlov, “A novel Internet of things-based plug-and-play multigas sensor for environmental monitoring,” *Transactions on Emerging Telecommunications Technologies*, vol 32, no. 6, Apr. 2020.

[258] Mauro A. A. da Cruz, Joel. J. P. C. Rodrigues, Gustavo F. A. Gomes, Pedro Almeida, Ricardo A. L. Rabelo, Neeraj Kumar, Shahid Mumtaz, “An IoT-Based Solution for Smart Parking,” in *Proceedings of First International Conference on Computing, Communications, and Cyber-Security (IC4S 2019)*, Chandigarh, India, 12-13 Oct. 2019, pp. 213–224.

[259] Sinara P. Medeiros, Joel J. P. C. Rodrigues, Mauro A. A. da Cruz, Ricardo A. L. Rabelo, Kashif Saleem, and Petar Solic, “Windows Monitoring and Control for Smart Homes

based on Internet of Things," in *2019 4th International Conference on Smart and Sustainable Technologies (SpliTech)*, Split, Croatia, 18-21 Jun. 2019, pp. 1–5.

[260] Kellow Pardini, Joel J. P. C. Rodrigues, Ousmane Diallo, Ashok K. Das, Victor H. C. de Albuquerque, and Sergei A. Kozlov, "A smart waste management solution geared towards citizens," *Sensors*, vol. 20, no. 8, p. 2380, Apr. 2020.

[261] Joel J. P. C. Rodrigues and Mauro A. A. da Cruz, "In: IoT, registry request of computer program in Brazil—RPC N BR 512018051862-1" National Institute of Telecommunications, Santa Rita do Sapucaí, Brazil, Rep. 1, Oct. 2018.

[262] "In-IoT — Bitbucket." [Online]. Available: https://bitbucket.org/In-IoT/. [Accessed: 24-Jan-2021].

[263] Robert Waszkowski, "Low-code platform for automating business processes in manufacturing," *IFAC-PapersOnLine*, vol. 52, no. 10, pp. 376–381, 2019.

[264] R. Sanchis, Ó. García-Perales, F. Fraile, and R. Poler, "Low-Code as Enabler of Digital Transformation in Manufacturing Industry," *Applied Sciences*, vol. 10, no. 1, p. 12, Dec. 2019.

[265] William Villegas-Ch., Joselin García-Ortiz, and Santiago Sánchez-Viteri, "Identification of the Factors That Influence University Learning with Low-Code/No-Code Artificial Intelligence Techniques," *Electronics*, vol. 10, no. 10, p. 1192, May 2021.

[266] José M. Sáez-López, Javier del Olmo-Muñoz, José A. González-Calero, and Ramón Cózar-Gutiérrez, "Exploring the Effect of Training in Visual Block Programming for Preservice Teachers," *Multimodal Technologies and Interaction*, vol. 4, no. 3, p. 65, Sep. 2020.

[267] Henrique Henriques, Hugo Lourenço, Vasco Amaral, and Miguel Goulão, "Improving the Developer Experience with a Low-Code Process Modelling Language," in *MODELS '18: ACM/IEEE 21th International Conference on Model Driven Engineering Languages and Systems*, New York, NY, USA, 14-19 Oct. 2018, pp. 200–210.

[268] Vadim Zaytsev, "Open challenges in incremental coverage of legacy software languages," in *SPLASH '17: Conference on Systems, Programming, Languages, and Applications: Software for Humanity*, Vancouver, BC, Canada, 22 Oct. 2018, pp. 1–6, doi: 10.1145/3167105.

[269] M. Völter, T. Stahl, J. Bettin, A. Haase, and S. Helsen, *Model-driven software development: technology, engineering, management*. Chichester, England: John Wiley & Sons, 2006.

[270] Gartner, "Magic Quadrant for Enterprise Low-Code Application Platforms," 2020.

[271] Marcus Woo, "The Rise of No/Low Code Software Development—No Experience Needed?," *Engineering*, vol. 6, no. 9, pp. 960–961, Sep. 2020.

[272] Felicien Ihirwe, Davide Di Ruscio, Silvia Mazzini, Pierluigi Pierini, and A. Pierantonio, "Low-code engineering for internet of things," in *MODELS '20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems*, Virtual Event Canada, 16-23 Oct. 2020, pp. 1–8, doi: 10.1145/3417990.3420208.

[273] Antonio Bucchiarone, Federico Ciccozzi, Leen Lambers, Alfonso Pierantonio, Mathias

Tichy, Massimo Tisi, Andreas Wortmann, and Vadim Zaytsev“What Is the Future of Modeling?,” *IEEE Software*, vol. 38, no. 2, pp. 119–127, Mar. 2021.

[274] Ricardo Martins, Filipe Caldeira, Filipe Sá, Maryam Abbasi, and Pedro Martins, “An overview on how to develop a low-code application using OutSystems,” in *2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)*, Bengaluru, India, 9-10 Oct. 2020, pp. 395–401.

[275] Brian Broll, Ákos Lédeczi, Péter Völgyesi, János Sallai, Alexia Carrillo, and Miklós Maróti, “A Visual Programming Environment for Learning Distributed Programming,” in *SIGCSE ’17: The 48th ACM Technical Symposium on Computer Science Education*, Seattle, WA, USA, 8-11 Mar. 2017, pp. 81–86.

[276] Apurvanand Sahay, Arsene Indamutsa, Davide Di Ruscio, and Alfonso Pierantonio, “Supporting the understanding and comparison of low-code development platforms,” in *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Portoroz, Slovenia, 26-28 Aug. 2020, pp. 171–178.

[277] Masoud Barati, Omer Rana, Ioan Petri, and George Theodorakopoulos, “GDPR Compliance Verification in Internet of Things,” *IEEE Access*, vol. 8, pp. 119697–119709, Jun. 2020.

[278] W. Stallings, “Handling of Personal Information and Deidentified, Aggregated, and Pseudonymized Information Under the California Consumer Privacy Act,” *IEEE Security & Privacy*, vol. 18, no. 1, pp. 61–64, Jan. 2020.

[279] Carlos H. T. Arteaga, Faiber B. Anacona, Kelly T. T. Ortega, and Oscar M. C. Rendon, “A Scaling Mechanism for an Evolved Packet Core Based on Network Functions Virtualization,” *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 779–792, Jun. 2020.

[280] R. Gouareb, V. Friderikos, and A. H. Aghvami, “Placement and Routing of VNFs for Horizontal Scaling,” *2019 26th International Conference on Telecommunications (ICT)*, Hanoi, Vietnam, 8-10 Apr. 2019, pp. 154–159.

[281] Faezeh Khorram, Jean-Marie Mottu, and Gerson Sunyé, “Challenges & opportunities in low-code testing,” in *MODELS ’20: ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems*, Virtual Event Canada, 16-23 Oct. 2020, pp. 1–10.

[282] C. Mouradian, S. Kianpisheh, and R. H. Glitho, “Application Component Placement in NFV-based Hybrid Cloud/Fog Systems,” in *2018 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, Washington, DC, USA, 25-27 Jun. 2018, pp. 25–30.

[283] António Manso, Célio G. Marques, Paulo Santos, Luís Lopes, and Raquel Guedes, “Algorithmi IDE-Integrated learning environment for the teaching and learning of algorithmics,” in *2019 International Symposium on Computers in Education (SIIE)*, Tomar, Portugal, 21-23 Nov. 2019, pp. 1-6.

[284] “GitHub - heitor-lassarote/iolp: Inatel Open Low-Code Platform.” [Online]. Available: https://github.com/heitor-lassarote/iolp. [Accessed: 20-Oct-2021].

[285] Vikas Hassija, Vinay Chamola, Vikas Saxena, Divyansh Jain, Pranav Goyal, and Biplab Sikdar, “A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures,” *IEEE Access*, vol. 7, pp. 82721–82743, Jun. 2019.

[286] Sherali Zeadally, Ashok K. Das, and Nicolas Sklavos, “Cryptographic technologies and protocol standards for Internet of Things,” *Internet of Things*, vol. 14, p. 100075, Jun. 2021.

[287] I. Farris, T. Taleb, Y. Khettab, and J. Song, “A Survey on Emerging SDN and NFV Security Mechanisms for IoT Systems,” *IEEE Communucations Surveys & Tutorials*, vol. 21, no. 1, pp. 812–837, 1st Quarter 2019.

[288] H. Kim, H. Lee, and H. Lim, “Performance of Packet Analysis between Observer and WireShark,” in *2020 22nd International Conference on Advanced Communication Technology (ICACT)*, Phoenix Park, South Korea, 16-19 Feb. 2020, pp. 268–271.

[289] M. A. Aladaileh, M. Anbar, I. H. Hasbullah, Y.-W. Chong, and Y. K. Sanjalawe, “Detection Techniques of Distributed Denial of Service Attacks on Software-Defined Networking Controller–A Review,” *IEEE Access*, vol. 8, pp. 143985–143995, Aug. 2020.

[290] M. Vigenesh and R. Santhosh, “An efficient stream region sink position analysis model for routing attack detection in mobile ad hoc networks,” *Computers & Electrical Engineering*, vol. 74, pp. 273–280, Mar. 2019.

[291] Sunghwan Kim, Seunghyun Yoon, Jargalsaikhan Narantuya, and Hyuk Lim, “Secure Collecting, Optimizing, and Deploying of Firewall Rules in Software-Defined Networks,” *IEEE Access*, vol. 8, pp. 15166–15177, Jan. 2020.

[292] Zhao Huang, Quan Wang, Yin Chen, and Xiaohong Jiang, “A Survey on Machine Learning Against Hardware Trojan Attacks: Recent Advances and Challenges,” *IEEE Access*, vol. 8, pp. 10796–10826, Jan. 2020.

[293] Tejasvi Alladi, Vinay Chamola, Biplab Sikdar, and Kim-Kwang R. Choo, “Consumer IoT: Security Vulnerability Case Studies and Solutions,” *IEEE Consumer Electronics Magazine*, vol. 9, no. 2, pp. 17–25, Mar. 2020.

[294] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas, “DDoS in the IoT: Mirai and Other Botnets,” *Computer*, vol. 50, no. 7, pp. 80–84, Jul. 2017.

[295] I.-G. Lee, K. Go, and J. H. Lee, “Battery Draining Attack and Defense against Power Saving Wireless LAN Devices,” *Sensors*, vol. 20, no. 7, p. 2043, Apr. 2020.

[296] S. Garcia, A. Parmisano, and M. J. Erquiaga, “IoT-23: A labeled dataset with malicious and benign IoT network traffic.” Zenodo, Jan. 20, 2020.

[297] Pedro M. S. Sanchez, José M. J. Valero, Alberto H. Celdran, Gérôme Bovet, Manuel G. Perez, and Gregorio M. Perez, “A Survey on Device Behavior Fingerprinting: Data Sources, Techniques, Application Scenarios, and Datasets,” *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 1048–1077, 2nd Quarter 2021.

[298] Kalupahana L. K. Sudheera, Dinil M. Divakaran, Rhishi P. Singh, and Mohan Gurusamy, “ADEPT: Detection and Identification of Correlated Attack Stages in IoT Networks,” *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6591–6607, Apr. 2021.

[299] Nour Moustafa and Jill Slay, “UNSW-NB15: a comprehensive data set for network

intrusion detection systems (UNSW-NB15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, Canberra, ACT, Australia, 10-12 Nov. 2015, pp. 1–6.

[300] "GitHub - kaysudheera/NSS_Mirai_Dataset: This dataset is captured from a Mirai type botnet attack on an emulated IoT network in OpenStack." [Online]. Available: https://github.com/kaysudheera/NSS_Mirai_Dataset. [Accessed: 20-Sep-2021].

[301] Mandira Hegde, Gilles Kepnang, Mashail Al Mazroei, Jeffrey S. Chavis, and Lanier Watkins, "Identification of Botnet Activity in IoT Network Traffic Using Machine Learning," in *2020 International Conference on Intelligent Data Science Technologies and Applications (IDSTA)*, Valencia, Spain, 19-22 Oct. 2020, pp. 21–27.

[302] Vibekananda Dutta, Michal Choraś, Marek Pawlicki, and Rafal Kozik, "A Deep Learning Ensemble for Network Anomaly and Cyber-Attack Detection," *Sensors*, vol. 20, no. 16, p. 4583, Aug. 2020.

[303] Robertas Damasevicius, Algimantas Venckauskas, Sarunas Grigaliunas, Jevgenijus Toldinas, Nerijus Morkevicius, Tautvydas Aleliunas, and Paulius Smuikys, "LITNET-2020: An Annotated Real-World Network Flow Dataset for Network Intrusion Detection," *Electronics*, vol. 9, no. 5, p. 800, May 2020.

[304] O. Barut, Y. Luo, T. Zhang, W. Li, and P. Li, "NetML: A Challenge for Network Traffic Analytics," Apr. 2020.

[305] Agathe Blaise, Mathieu Bouet, Vania Conan, and Stefano Secci, "Botnet Fingerprinting: A Frequency Distributions Scheme for Lightweight Bot Detection," *IEEE Transactions on Network and Service Management*, vol. 17, no. 3, pp. 1701–1714, Sep. 2020.

[306] S. García, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *Computers & Security*, vol. 45, pp. 100–123, Sep. 2014.

[307] Chen Wang, Chengyuan Deng, and Suzhen Wang, "Imbalance-XGBoost: leveraging weighted and focal losses for binary label-imbalanced classification with XGBoost," *Pattern Recognition Letters*, vol. 136, pp. 190–197, Aug. 2020.

[308] Lucas M. Policarpo, Diórgenes E. da Silveira, Rodrigo R. Righi, Rodolfo A. Stoffel, Cristiano A. da Costa, Jorge L. V. Barbosa, Rodrigo Scorsatto, Tanuj Arcot, "Machine learning through the lens of e-commerce initiatives: An up-to-date systematic literature review," *Computer Science Review*, vol. 41, p. 100414, Aug. 2021.

[309] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu, "Machine Learning Testing: Survey, Landscapes and Horizons," *IEEE Transactions on Software Engineering*, pp. 1–1, 2020.

[310] Hidetaka Taniguchi, Hiroshi Sato, and Tomohiro Shirakawa, "A machine learning model with human cognitive biases capable of learning from small and biased datasets," *Scientific Reports*, vol. 8, no. 1, p. 7397, May 2018.

[311] Shihab E. Saad and J. Yang, "Twitter Sentiment Analysis Based on Ordinal Regression," *IEEE Access*, vol. 7, pp. 163677–163685, Nov. 2019.

[312] Fenglian Li, Xueying Zhang, Xiqian Zhang, Chunley Du, Yue Xu, and Yu-Chu Tian, "Cost-sensitive and hybrid-attribute measure multi-decision tree over imbalanced data

sets," *Information Sciences*, vol. 422, pp. 242–256, Jan. 2018.

[313] Hyunsoo Choi, Siwon Kim, Jungeun Oh, Jeeeun Yoon, Jungah Park, Changho Yun, and Sungroh Yoon, "XGBoost-Based Instantaneous Drowsiness Detection Framework Using Multitaper Spectral Information of Electroencephalography," in *BCB '18: 9th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics*, 2018, Washington, DC, USA, 29 Aug. - 1 Sep. 2019, pp. 111–121.

[314] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, California, USA, 13-17 Aug. 2016, pp. 785–794.