UNIVERSIDADE D
COIMBRA

José Nuno da Cruz Faria

# WIRELESS IOT SMART BED SYSTEM

**Dissertação no âmbito do Mestrado Integrado em Engenharia Física, ramo de Instrumentação orientada pelo Doutor David Bina Siassipour Portugal, coorientada pelo Prof. Doutor Mahmoud Tavakoli e apresentada ao Departamento de Física da Faculdade de Ciências e Tecnologia da Universidade de Coimbra**

Fevereiro de 2022


FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

# Wireless IoT Smart Bed System

**José Nuno da Cruz Faria**

Coimbra, February 2022


# Wireless IoT Smart Bed System

**Supervisor:**

Doctor David B. S. Portugal

**Co-Supervisor:**

Prof. Doctor Mahmoud Tavakoli

**Jury:**

Prof. Doctor António Miguel Lino Santos Morgado

Prof. Doctor Paulo José Monteiro Peixoto

Doctor David Bina Siassipour Portugal

Dissertation submitted in partial fulfillment for the degree of Master of Science in Engineering Physics.

Coimbra, February 2022

# Acknowledgments

During these last years, several people have been paramount for the conclusion of this thesis, and as such, I would like to acknowledge all of those who helped me in some way or another to achieve this.

First and foremost, I want to thank my family, for their love and care, who supported and helped me throughout all these years, and without whom I would never be able to continuously push forward.

I also wish to extend my deepest gratitude to my supervisors, Prof. Mahmoud Tavakoli and in particular Dr. David Portugal, for his guidance, his continuous support, and for giving me an opportunity to undertake this challenge and so much more; I will be forever grateful.

Finally, to my closest friends, Edgar, Idálio, Daniel, Afonso, Gabriel and Brito for putting up with me all of this time, which I know to be a very troublesome task at times, and being a helping hand on this journey, thank you.

# Resumo

Nos últimos anos, em particular com o desenvolvimento da pandemia COVID-19, temos observado uma transformação digital continua dos cuidados de saúde. Uma tecnologia em específico destaca-se, revelando grande potencial para revolucionar o paradigma atual dos cuidados de saúde – a Internet das Coisas (IoT).

O trabalho desta dissertação consiste no desenvolvimento de uma infrastrutura IoT que interliga sensores inteligentes vestíveis a um sistema de informação de saúde utilizado por inúmeros hospitais nacionais. Em particular, é efetuada uma avaliação de diferentes placas de hardware IoT comuns para determinar que placa será utilizada para a infraestrutura de aquisição de dados, além de uma análise extensiva da comunicação Bluetooth Low Energy (BLE) para avaliar os dispositivos utilizados para efetuar a comunicação BLE. No âmbito deste trabalho foi também proposta uma especificação para comunicação Message Queuing Telemetry Transport (MQTT), tal como o desenvolvimento de uma Interface de Programação de Aplicações (API) usando o protocolo Fast Healthcare Interoperability Resources (FHIR), que é um dos protocolos mais utilizados para a transmissão de informação de saúde no ambiente de software clínico, para integrar a infraestrutura no sistema de informação de saúde, juntamente com uma arquitetura de serviços para o servidor *edge*, desenhado com segurança e privacidade em mente.

A plataforma desenvolvida é validada através de um ensaio em instalações hospitalares e testes em ambiente controlado, nos quais a infraestrutura mostrou uso eficiente de recursos, disponibilidade máxima e boa segurança, fornecendo uma base sólida para continuar o desenvolvimento e expandir as funcionalidades existentes do sistema.

**Palavras-chave:** Internet das Coisas; FHIR; MQTT; Bluetooth Low Energy; Cuidados de saúde; Sensores vestíveis.

# Abstract

In the past years, especially with the COVID-19 pandemic, we have observed the continuous digital transformation of healthcare. But there is one technology in particular that shows potential to revolutionize the current healthcare paradigm – the Internet of Things (IoT).

This work focuses on the development of a secure Internet of Things infrastructure which connects wireless, wearable, and intelligent sensors to a health information system used by multiple national hospitals. In particular, an extensive benchmarking of common IoT hardware platforms to serve as IoT acquisition nodes is performed, and an analysis of Bluetooth Low Energy (BLE) communication protocol to evaluate the adequacy of the devices used to implement it. Additionally, a specification for Message Queuing Telemetry Transport (MQTT) communication is also proposed, as well as the development of an Application Programming Interface (API) using Fast Healthcare Interoperability Resources (FHIR), an open standard widely used in health informatics for exchanging electronic health records, to integrate the IoT infrastructure with the health information system, coupled with an efficient service architecture for the edge server, designed with security and privacy in mind.

The work is validated both through trials within hospital facilities and controlled laboratory tests, in which the infrastructure shows efficient resource usage, maximum availability and security, providing solid foundations for future work to build upon.

**Keywords:** Internet of Things; FHIR; MQTT; Bluetooth Low Energy; Healthcare; Wearable sensors.

# Contents

# List of Acronyms

| | |
|---|---|
| **API** | Application Programming Interface |
| **ATT** | Attribute Protocol |
| **BLE** | Bluetooth Low Energy |
| **CoAP** | Constrained Application Protocol |
| **CPU** | Central Processing Unit |
| **CRUD** | Create, Read, Update and Delete |
| **DLE** | Data Length Extension |
| **ECG** | Electrocardiogram |
| **EHR** | Eletronic Health Record |
| **EPC/RFID** | EPCglobal Gen2 RFID |
| **FHIR** | Fast Healthcare Interoperability Resources |
| **GAP** | Generic Access Profile |
| **GATT** | Generic Attribute Profile |
| **HCI** | Host Controller Interface |
| **HIS** | Health Information System |
| **HTTP** | Hypertext Transfer Protocol |
| **IMU** | Inertial Measurement Unit |
| **IP** | Internet Protocol |
| **IPC** | Interprocess communication |

| | |
|---|---|
| **ISR** | Institute of Systems and Robotics |
| **IT** | Information Technology |
| **IoT** | Internet of Things |
| **JSON** | JavaScript Object Notation |
| **L2CAP** | Logical Link Control and Adaptation Protocol |
| **LL** | Link Layer |
| **MQTT** | Message Queuing Telemetry Transport |
| **MTU** | Maximum Transmission Unit |
| **NTP** | Network Time Protocol |
| **OSI** | Open System Interconnection |
| **PDU** | Protocol Data Unit |
| **PHY** | Physical Layer |
| **PPG** | Photoplethysmography |
| **RAM** | Random Access Memory |
| **RBAC** | Role-based access control |
| **RDBMS** | Relational Database Management System |
| **RFID** | Radio-frequency Identification |
| **SBC** | Single Board Computer |
| **SIG** | Special Interest Group |
| **SMP** | Security Manager |
| **SQL** | Structured Query Language |
| **SoC** | System on a Chip |
| **TCP** | Transmission Control Protocol |
| **TLS** | Transport Layer Security |

| | |
|---|---|
| **UI** | User Interface |
| **UUID** | Universally Unique Identifier |
| **WBAN** | Wireless Body Area Network |
| **WoW** | Wireless biOmonitoring stickers and smart bed architecture: toWards Untethered Patients |
| **XML** | Extensible Markup Language |

# List of Figures

# List of Tables

# 1 Introduction

## 1.1 Context

Due to all the technological and healthcare advances over the last years, we have observed a steady increase of life expectancy. With the growing aging population a rise of chronic illnesses can be noticed, which places significant strain on modern healthcare systems due to their limited resources [1, 2] - both human and material like medical equipment, hospital beds, etc. This is an evermore pressing concern, particularly with the recent Covid-19 epidemic, that is testing the limits of current healthcare systems.

In an effort to counter this, many countries and organizations are currently promoting the shift towards digital healthcare through several funding programs, such as European Union [3] and World Health Organization initiatives [4]. The usage of digital technologies in health has the potential to radically change how healthcare is delivered, by enhancing the efficiency and cost-effectiveness of care, and enabling new business models for service providers [4].

In particular, there is one paradigm which has stood out, having potential to fulfill this vision for digital health – **Internet of Things (IoT)**. IoT has been used to revolutionize different industries, such as smart grids [5], oil and gas industry [6] and many more. The basic concept of IoT is enabling processing capability and connectivity of "physical objects" or groups of these, allowing them to be capable of connecting with other devices and exchange data through the Internet [7]. Today, hospitalized patients need to be wired to various measurement instruments when continuous biomonitoring is required. IoT is not only capable of challenging this restriction, detaching patients from their beds and restoring their much-needed comfort (perhaps even allowing them to return to their own homes) [8], but also provide value to the various stakeholders in the healthcare system with the automatization of processes, continuous and remote monitoring, clinical decision support, etc.

However, there are still many challenges to tackle before deploying such technologies in a medical environment. In particular, interoperability, security, and privacy are challenges which are recurrently identified in the literature.

Any device that is exposed to the Internet is a possible security liability, and thus the development of efficient security measures is crucial to ensure that data remains private.

Moreover, the adoption of these novel systems can be often met with much objection from the clinical staff due to their mistrust of technology [9]. To facilitate their deployment in hospitals, these need to be integrated easily in existing Health Information Systems (HIS).

## 1.2 System Requirements

Previous work by researchers at the Institute of Systems and Robotics (ISR) has resulted in the development of innovative wearable devices, designated *Biostickers*, which are electronic patches equipped with sensors that gather the patient's physiological signals and communicate wirelessly [10].

**The main objective of this dissertation work** is the development and validation of an IoT architecture capable of integrating the data that is acquired by the *Biostickers* into an existing HIS, in the context of the WoW R&D project[1]. This system serves as the "backbone" of the entire Information Technology (IT) system for the project, connecting these *Biostickers* to the HIS while addressing crucial issues as described previously.

The requirements for the system, and their priorities based on a MoSCoW prioritization analysis [11], are the following:

- **Must:**
  - The system must be able to communicate with the HIS, i.e. capable of processing incoming requests and transmit necessary data.
  - All communications within the system must be secure, and any sensitive data cannot be accessed by unauthorized third parties.
  - The system must be able to function for long periods of time without continuous support or maintenance.
  - The system must be non-invasive and intuitive.

[1]WoW – A step towards domiciliary hospitalization: `https://inovglintt.com/financiamento/wow/`

- **Should:**
  - The system should adopt international standards for exchanging information.
- **Could:**
  - With long-term monitoring, large amounts of information could be gathered to create valuable datasets that can be used to improve patient monitoring.
- **Would:**
  - For long-term patient monitoring, the system would be capable of identifying anomalies / biomarkers in patients' biosignals.

## 1.3 Dissertation Structure

This document is organized into different sections. The first chapter provides an introduction to the theme of the dissertation, discussing the context and motivation behind the work carried out. In the second chapter a brief overview into IoT infrastructures and its healthcare applications is shown, along with a statement of the contributions of this work. The third chapter focuses on hardware analysis for one of the IoT system components, the *Smart box*. The fourth chapter describes the service architecture within another system component, the *Smart Gateway*, which is validated experimentally and analyzed in the fifth chapter. Finally, in the sixth and final chapter, a reflection upon the work is performed, concluding with a discussion of completed objectives and on future work.

# 2 State of the Art

In this chapter a survey of pervasive healthcare applications is presented. In order to gain a greater understanding of the building blocks of a typical Internet of Things (IoT) system, a reference model for IoT systems is also discussed. With this knowledge, a comparative analysis of similar works in the literature is presented, identifying the strengths and weaknesses of each approach. The chapter is concluded by stating the contributions that this work proposes to achieve.

## 2.1 Internet of Things

### 2.1.1 Fundamentals of IoT

Internet of Things (or IoT) is an emerging communication paradigm, often hailed as the driver of the Fourth Industrial Revolution [12].

The definition of this concept has evolved over time with the development of other technologies such as data analytics, embedded systems, sensors, etc. Fundamentally, it can be described as the following [13]:

> IoT addresses the development of networks of smart devices that exchange and process information through Machine-to-Machine (M2M) communications, usually based on the Internet Protocol (IP).

This technology enables ubiquitous systems to gather remarkable amounts of information regarding the surrounding environment, which can later be turned into insight through the usage of data fusion and data analytics tools, like Machine Learning.

In the specific context of healthcare, this technology can provide many benefits as it enables remote and continuous health monitoring [14, 15, 16]. It allows non-critical patients

to be monitored from the comfort of their own houses, rather than in hospitals or clinics, reducing the strain on scarce hospital resources such as health professionals or beds. This is particularly beneficial to those who live in rural areas, with limited access to healthcare services. It enables elderly people and those with chronic diseases to have greater control over their own health, thus allowing them to live more independently. Moreover, with the automatization of medical procedures, these systems can make healthcare infrastructures more efficient and therefore lower the costs of healthcare [17, 18]. Particularly, in the realm of clinical research, by analyzing the data collected by these ubiquitous systems, it may be possible to find new relationships between certain pathologies and different physiological signals, such as variations in body temperature or heart rate [19]. These correlations, commonly referred to as biomarkers, can be used by these systems to assist clinical decisions, enabling novel predictive, prognostic, and diagnostic processes in healthcare.

## 2.2 A Reference Model for Pervasive Healthcare Applications

Reference models provide an abstract framework for designing systems and a set of commonly recommended practices for the application domain. It serves as a starting point in the design process, enabling the comprehension of complex systems by breaking them down into simple and distinct functional layers, while also defining some common terminology used in its domain.

In 2014, the IoT World Forum (IoTWF) architectural committee published an IoT architectural reference model, composed by seven layers as shown in Figure 2.1. This model [20] provides a simple and clean functional view into the different components of an IoT system, without narrowing the scope or locality of its components. While this model can be used to generically develop IoT systems for any industry (*e.g.* from agriculture to smart cities), in the context of the dissertation our focus will remain on pervasive healthcare applications and related technologies.

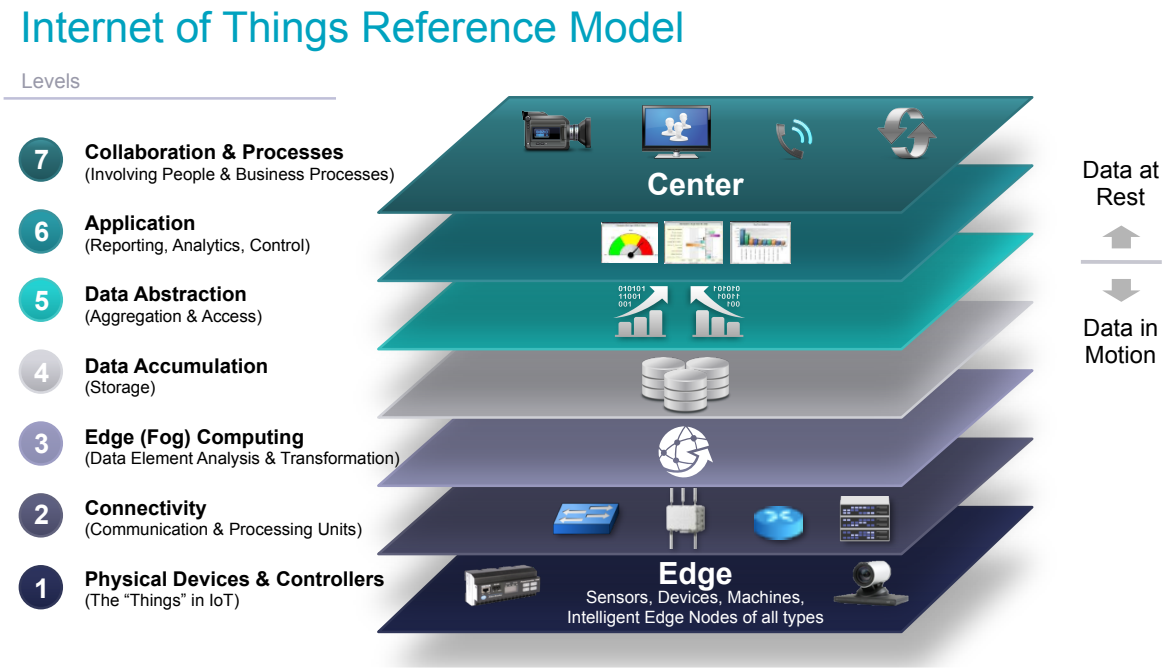


Figure 2.1: IoT reference model published by IoTWF. Source: [20].

### 2.2.1 Layer 1: Physical Devices and Controllers

The first layer of the model [20] corresponds to the physical devices and controller layer. This layer houses the "things" in the Internet of Things: the endpoint devices composed of sensors and actuators that perceive and interact with the physical world. Through those interactions, the devices generate data, which is then sent across the network for analysis and storage.

Wearable, wireless, and non-intrusive devices are viewed as one of the key components of IoT-based healthcare systems [13]. In recent years there has been remarkable progress on the development of wearable devices, driven by recent technological breakthroughs in the miniaturization of sensors and microfabrication processes [18, 17]. These devices allow patients to be monitored while retaining their mobility, increasing the comfort of these users. The drawback of this approach lies on the restrictions imposed on the devices. Due to the nature of this technology, most of these units require a portable energy source, which implies reduced memory, computation, and connectivity capabilities in order to minimize energy consumption and maximize their lifetime. Shorter lifetimes translate into higher maintenance costs, as these devices need to be replaced more often.

Another point to consider is the data requirements of the system, namely how much data is generated and which type of data is transmitted by each device. Some applications can

include a single temperature sensor or heart rate sensor, while more complex systems can include pulse oximetry, electrocardiogram (ECG), respiration rate sensors, etc. [15]. From the literature [15, 21, 18, 22], the sensors used in these devices can be classified into three distinct categories based on the signals that can be extracted from them, as shown in Table 2.1:

- **Physiological Sensors**: used for evaluating the patients' health condition.
- **Activity / Motion sensors**: used for detecting fall events, determining the patients location and the travelled distance, estimating the patients body posture, etc.
- **Environmental Sensors**: used for assessing environment conditions and possible hazards, *e.g.* gas leaks in a patients home or an industrial workplace [21].

Table 2.1: Type of sensors commonly used in pervasive healthcare applications. Adapted from [22].

| **Sensor Categories** | **Examples** |
|---|---|
| Vital Signs Monitoring | Blood Pressure, ECG, PPG, Body Temperature, Respiratory Rate, Galvanic Skin Response, Pulse Oximetry, Glucose Level Sensors |
| Activity Monitoring | Accelerometer, Gyroscope, Magnetometer |
| Environmental Monitoring | Air Temperature, Barometer, Humidity, Gas Sensors |

### 2.2.2 Layer 2: Connectivity

The second layer of the model focuses on connectivity, on linking the different components of the system, ensuring reliable and timely data transmissions. This includes all communications within the system, which can be divided into two categories: communications within the local network (*e.g.* between edge nodes and the gateway devices), and communications between the edge of the local network (*e.g.* gateway devices) and the central server.

### Communication Protocols

Technology is designed with particular use cases in mind, built to fulfill a certain need which other similar technologies fail to meet [20]. As such, each technology will be different, having advantages and disadvantages depending on its usage. For instance, short range wireless protocols are, by definition, limited by transmission range, but longer range protocols have in general a higher energy consumption, which may become unviable for networks with highly constrained devices. Some protocols communicate within certain frequency bands, some of which may require special licenses. Using licensed frequency bands can provide a better performance as it ensures greater reliability since the network operator grants you exclusivity of frequency spectrum within certain areas but may be too costly.

From the literature, a set of key requirements that drive the decision of the communication protocols can be identified [13, 17, 18]:

- **Energy consumption**: For networks composed of energy constrained devices, the communication protocol should be lightweight and energy efficient in order to maximize the devices' lifetime.
- **Latency**: Certain applications deal with time critical events, for example the detection of health emergencies [17]. In these cases, any delays in the communications can cause great detriment to the patients well-being, making it crucial to minimize them.
- **Reliability**: Depending on the critical nature of the data that is being communicated, the network stack may need to implement processes such as error-detection, retransmission or handshakes in the communications to ensure more robust transmissions, *e.g.* as implemented in TCP/IP based protocols. Generally, these features come at the cost of greater latency. Therefore, when choosing the communication protocol, a balance must be found between reliability and latency.
- **Security**: Security is one of the most important requirements of any system, but this is especially true for healthcare applications. Due to the sensitive nature of the information, it is crucial to secure it from malicious actors. Communication protocols must implement security mechanisms, such as encryption or data integrity verifications, that ensure the transmissions are not compromised in transit, thus denying third parties the ability to snoop or tamper the transmissions. This issue is studied in depth in [23].

- **Interoperability**: To ensure the interoperability of intrinsically different modules of the system it is imperative to choose protocols that are widely accepted and supported in the application domain. This also contributes to the longevity and maintenance of the system, as these will most likely remain supported for longer time periods.

- **Range**: The communication protocol must ensure that the devices can communicate within the required transmission range.

- **Scalability**: These systems may contain an enormous amount of devices, which must be uniquely identified. The communication protocol must ensure that every device in the network is addressable, and that performance is not severely impacted by the addition of new devices.

- **Throughput**: The communication protocol should ensure that there is enough bandwidth to handle all communications within the designated transmission range. Even within similar technologies, this can vary wildly with transmission range [24].

Regarding communications within the local network, these generally have short transmission ranges [13]. Wearable devices can be often arranged in networks, aptly designated Wireless Body Area Network (WBAN). The most widely adopted protocol is Bluetooth Low Energy (BLE), a low-energy version of the classic Bluetooth protocol [14, 21, 15]. ZigBee and Radio-frequency Identification (RFID) are also used in many systems in this domain, particularly in asset tracking oriented applications [25, 17, 18]. Despite the absence of a universal specification for RFID, the most widely used standard is the EPCglobal Gen2 RFID [26]. For the sake of simplicity, EPC/RFID is used to designate it.

Out of the abovementioned technologies, BLE offers greater throughput, better security, and nearly the lowest energy consumption [27]. Table 2.2 shows a comparison between the different protocols.

Table 2.2: Overview of the most common communication protocols used within short range. Adapted from [13].

| | **BLE** | **EPC/RFID** | **ZigBee** |
|---|---|---|---|
| **Band of operation** | 2.4 GHz | LF, HF, UHF, EHF | 2.4 GHz |
| **Communication** | Bidirectional | Unidirectional (Bidirectional for Active tags) | Bidirectional |
| **Topology** | Point-to-Point, Piconet, Broadcast, Mesh | Point-to-Point | Mesh |
| **Range** | <100 m | <10 m, (100 m for Active tags) | 20 m |
| **Data rate (Typical)** | 1Mbps | 40kbps | 250kbps |
| **IP Stack** | ✗ | ✗ | ✓ |
| **Security Features** | AES-128, Secure pairing prior to key exchange | ✗ | AES-128 (Optional), Network key shared across network, Optional link key to secure application layer communications |

Regarding communications between the gateway devices and the central server, most researchers try to make use of existing infrastructure in order to facilitate the deployment of new systems. This means, that the communications between the gateway devices and the central server are often done through generic IP-based protocols such as Wi-Fi and Ethernet [18, 25, 15, 17].

### Application Protocols

So far the underlying networking technologies that link the devices in system have been discussed. But according to the OSI Model, many of these technologies do not define the application layer: how the devices communicate with each other, how the data is formatted, if there is a hierarchy within the network, etc. When considering networks com-

posed of constrained devices, generic web-based protocols such as Hypertext Transfer Protocol (HTTP) may not be adequate for IoT applications, which prompts the development of novel lightweight messaging protocols suited for these systems. The most used application layer protocols in IoT systems are Message Queuing Telemetry Transport (MQTT) and Constrained Application Protocol (CoAP). Table 2.3 overviews these widely used protocols.

Table 2.3: Comparison between CoAP and MQTT protocols. Adapted from [24].

| | **MQTT** | **CoAP** |
|---|---|---|
| **Transport protocol** | TCP/IP | UDP/IP |
| **Messaging pattern** | Publish/Subscribe (asynchronous) | Request-Response (synchronous) |
| **Communication model** | Many-to-many | One-to-one |
| **Security** | SSL/TLS (Optional) | DTLS |
| **Strengths** | TCP and Quality of Service (QoS), robust communications, easier to implement | Better for lossy networks, lower latency |
| **Weaknesses** | Higher overhead and energy consumption than CoAP | Not as reliable and less supported than MQTT |

In [28], the authors compare these two protocols in greater length, along with the more commonly used web-based protocol Hypertext Transfer Protocol (HTTP). They analyze the latency in the communications (from the edge devices to remote servers) and the RAM usage in the devices for each protocol and for different data sources (respiration rate, oxygen saturation and heart rate signals) and found that CoAP presented the best overall performance. Nonetheless, all protocols had very low latencies (less than 1.5s) and low memory usage.

The authors indicate that MQTT might be more suitable when considering a certain messaging pattern — the "Publish/Subscribe" model. In this model, the devices that send messages, called "publishers", communicate them to an intermediary message broker, through a "message topic" (also called logical channels). The devices that wish to receive messages, called "subscribers", can subscribe to these topics by requesting it to the message broker.

Whenever a publisher sends a message, the broker broadcasts it to all devices that have subscribed to the selected topic. Figure 2.2 shows a diagram of MQTT of two clients (A and B) connecting to the MQTT broker, and where client A transmits two messages to the topic "client/b" while client B subscribes to that topic. In MQTT, a client subscribed to a topic receives a message whenever an authorized client (including itself) publishes a message on that topic.

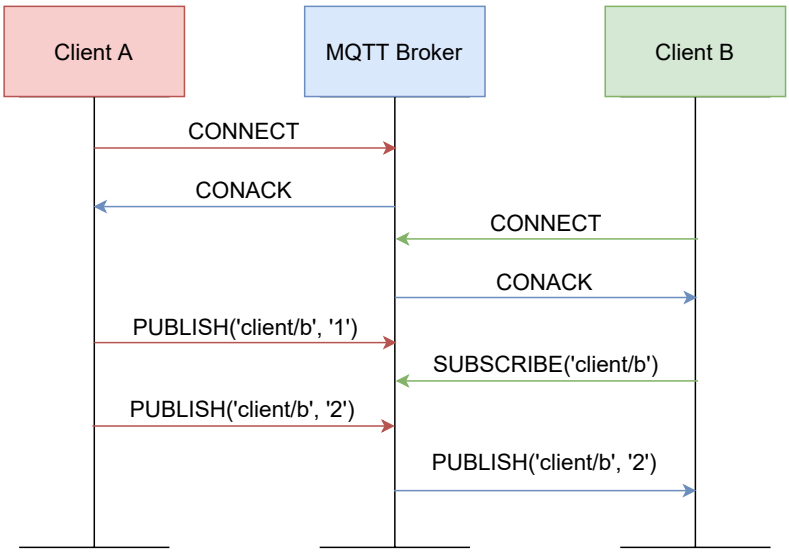


Figure 2.2: Diagram of a MQTT message sequence.

CoAP uses a different messaging pattern, called "Request-Response". In this paradigm, a device sends a request to receive certain data and the second responds to this request. In CoAP, the requests are delivered asynchronously, and so the response messages must contain a "token" value so that the client can identify which request resulted in this response. Figure 2.3 shows a diagram of the communication, where a client makes a GET request to the CoAP server to retrieve information.

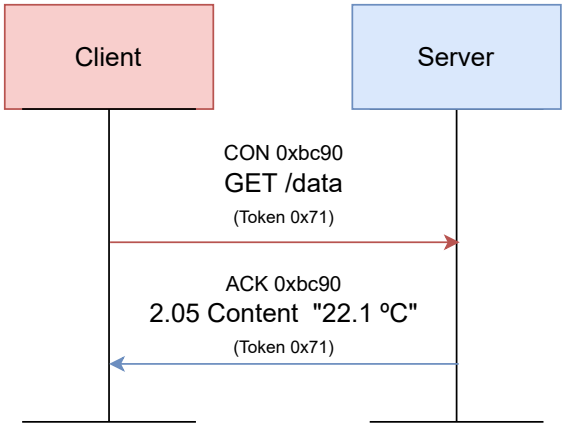


Figure 2.3: Diagram of a CoAP message sequence.

### 2.2.3 Layer 3: Edge (Fog) Computing

IoT systems may have hundreds or even thousands of sensors generating data multiple times per second, 24 hours per day, which may require an unsustainable amount of network and computing resources. Moreover, certain applications are time critical, where delays in communication can be very detrimental. To minimize these effects, it is crucial to initiate data processing as close to the edge of the network as possible. This paradigm is usually referred to as edge computing, when the data processing occurs at the endpoint devices, or fog computing, when it happens at the edge of the local network, *e.g.* in gateway devices.

The third layer of the model defines how the system prepares the data for storage and higher level processing for the next layers. However, the endpoint or gateway devices often have limited computing capabilities, so the data processing is generally focused on preprocessing the data in real-time and handling more time critical events. More demanding and thorough data analysis is usually left to the central server.

The different processes applied at this stage can be summarized into four distinct categories:

- **Filtering**: Assessing if the data should be processed at a higher level.
- **Formatting**: Reformatting data to ensure consistent formats for higher-level processing.
- **Cleaning**: Reducing data to minimize the impact of data on the network and higher level processing systems.
- **Evaluation**: Determining whether data represents a threshold or alert. This is especially relevant for applications that deal with time critical events as seen in the previous section.

### 2.2.4 Layer 4: Data Accumulation

The data that is generated by the edge devices is propagated through the system, and eventually reaches the central server. Up to this point, the model is event driven. However, most applications cannot make use of the data at the rate it is generated [24]. The Data Accumulation layer details how the system captures the data and stores it, so applications can make use of it when needed, thus transiting from event to query-based processing.

As the devices continuously generate data, the system will require more and more resources in order to process and store all of this information, raising some concerns regarding how the data can be managed. In [14], the authors propose the usage of cloud platforms as a solution to these problems. This is made possible due to the elasticity in allocating, swiftly and inexpensively, computing and storage resources on-demand, adjusting itself to the needs of each application. In general, three distinct types of cloud services can be identified:

- **Infrastructure as Service (IaaS)**: Provides control over the remote machine (composed of virtual or dedicated hardware), operative system and middleware. This approach gives system designers the highest level of flexibility over the infrastructure, but requires more maintenance.
- **Platform as a Service (PaaS)**: Provides a simple framework for developing applications, where the service provider manages the underlying infrastructure issues such as software updates and hardware maintenance.
- **Software as a Service (SaaS)**: Provides the finished applications to be used by the end users, in this case health workers, that enable them to work. A simple example is a web-based email service, such as Gmail or Microsoft Outlook.

Figure 2.4 shows the differences between these cloud offerings and on-premise deployments.

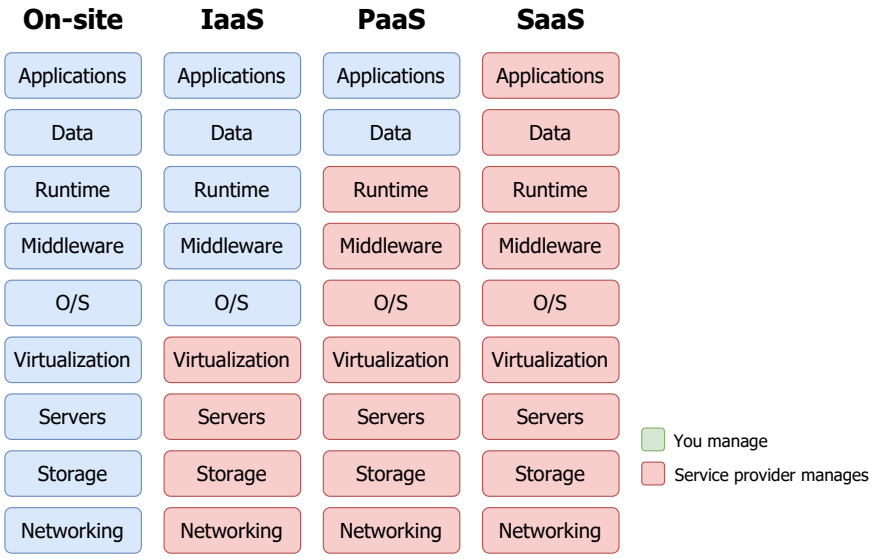


Figure 2.4: Differences between the cloud offerings and on-premise solutions. Source: [29]

Nonetheless, security and privacy remain as key concerns for the implementation of cloud-based solutions. The information must remain accessible to authorized parties such as healthcare providers, but the patients health data has to be kept private. To solve this, there are

two commonly adopted features in the literature: access control policies and data encryption [13]. Access control policies define who can access the data, by authenticating them (validating the identity of the user) and by authorizing them (ensuring that the user has permissions to perform a given operation). Data encryption ensures that, even if the data is leaked, it is still unreadable to third parties, and therefore sensitive information remains secure and private.

Regarding storage solutions, most research works use traditional relational databases (RDBMS) as the means of storing data [25, 15, 17, 18]. These are often referred to simply SQL or Structured Query Language (SQL) databases, which is the language used to interact with the database.

However, since the data in IoT can very heterogenous and unstructured, the authors of [30] propose using NoSQL databases. NoSQL or "not only SQL" describes a class of database systems that can support – and are more optimized for – storing semi-structured and unstructured data. NoSQL data stores typically outperform SQL databases as the data increases in volume [31]. Table 2.4 illustrates the differences between these two technologies.

Table 2.4: Comparison between SQL and NoSQL database technologies.

| | **SQL** | **NoSQL** |
|---|---|---|
| **Type of database** | Relational | Non-relational |
| **Database model** | Table-based database | Document-based databases, Key-value stores, graph stores, wide column stores |
| **Data type** | Appropriate for structured data | Appropriate for unstructured or semi-structured data |
| **Schema** | Strict schema | Dynamic schema |
| **Query** | Uses Standard Query Language (SQL), appropriate for complex query operations | No standard query language |
| **Scalability** | Vertical | Horizontal |
| **Performance** | Generally lower than NoSQL systems | Optimized for large datasets |

### 2.2.5 Layer 5: Data Abstraction

The previous layer defines how the system captures the information. In some cases, the collection of data may require the development of multiple concurrent storage solutions, each using different technologies, resulting in a very complex environment. The purpose of this layer is to define services that simplify how the applications access the data, to reconcile the different data stores and ensure the information is complete and consistent [20]. Applications can then interact with these databases through interfaces exposed by these services, designated Application Programming Interfaces (APIs).

An API is a computing interface that defines a set of rules that "explain how computers and applications communicate with one another", acting as an intermediary between these different components [32]. It defines which operations can be performed, how to request them, which are the accepted data types, etc. In this case, it decouples applications from the storage solutions, by encapsulating their functionality behind the interface. This ensures the modularity of the system as the applications become independent of whichever technologies are used in the data stores.

Understanding what and how information is shared within the healthcare domain is fundamental. As patients continuously circulate through the healthcare ecosystem, their health information must be available, discoverable and understandable to different entities (hospitals, laboratories, pharmacies, etc.). This prompts the digitization of medical files and the development of standards for exchanging these records instantly and securely to authorized parties [33], which are called Eletronic Health Records (EHRs). EHRs are the digital equivalent of a patient's paper-chart, containing the patient's full medical history: previous diagnoses, treatment plans, test results, known allergies, among other details.

One of the most prominent standards for exchanging EHRs is Fast Healthcare Interoperability Resources (FHIR). FHIR is a standard developed by Health Level Seven International (HL7), which is a non-profit organization involved in the development of international healthcare informatics for over 20 years. FHIR builds upon previous data format standards like HL7 v2 and HL7 v3, and is becoming widely adopted within the healthcare industry [34]. This standard defines a lightweight RESTful framework using common data formats, like JSON and XML, so it can be readily integrated into lightweight web services, thus underlining its suitability for web-based platforms [35].

### 2.2.6 Layer 6: Application

The sixth layer corresponds to the application layer, where the system ingests the captured data, analyzes it and delivers the value to the end users. Users can then interact with the system through an User Interface (UI), which provide different functionalities depending on the application. Some may show simple reports regarding the collected data [14, 15], and others may allow users to monitor and have greater control over the different components of the system.

As seen in an earlier section, Table 2.1 shows what kind of information is generally acquired in IoT healthcare applications. Using artificial intelligence, it is also possible to correlate all of this information to guide the clinical decisions from healthcare providers [35, 36, 37].

### 2.2.7 Layer 7: Collaboration and Processes

The information that is created by the IoT systems yields little value unless it prompts action, which requires integrating people and business processes (seventh layer). The purpose of these systems is to empower people to work better and more efficiently by providing valuable insight at the right time. To do this, people must be able to communicate and collaborate, which often requires multiple steps and transcends multiple applications [20]. However, this component of the system is beyond the scope of this work and thus it is not discussed further.

## 2.3 Survey on IoT Applications for Healthcare

The architecture of IoT systems has been thoroughly discussed, but so far no specific implementations have been referenced. This section presents an overview of IoT connected healthcare applications described in the literature, highlighting each of their strengths and weaknesses.

In [25], one of the first IoT applications for healthcare is described. The authors propose a real-time locating system (RTLS) using RFID tags called RFIDLocator. These tags are placed in hospital equipment, staff, patients and medical files and by using RFID readers placed in strategic locations around the hospital (*e.g.* entrance of rooms, handheld readers), it is possible to track the location of each object. When a RFID reader detects a RFID

tag it communicates this information, using Wi-Fi, to a central server which stores it in a MySQL database. Healthcare workers can then view this information through a web application, which contains a location history of the tagged object. The authors show how RTLS systems can mitigate the risks of patient misidentification, loss or theft of assets and even drug counterfeiting. However, in this article, security and privacy issues are not discussed. Although not stated explicitly, communications between the RFID tags and the RFID readers are assumed to be unencrypted, which means "unethical individuals could snoop on people and surreptitiously collect data (...) without their knowledge", even after leaving the hospital if the tags are not removed. This raises serious privacy concerns, as the tags could contain private information that can be detrimental to the patients if revealed.

In [18], the authors propose a RTLS system that also monitors the patient's vital signs, using a small wristband which holds a low power device equipped with temperature, photoplethysmography (PPG), used to obtain the heart rate, and accelerometer sensors, used for detecting fall events. The system can also detect with 70% accuracy if the patient has fallen, sending an immediate message to the gateway, which will later alert the clinical staff to the emergency. The authors ran a pilot test within hospital premises which was well-received by the clinical staff who praised the system for its intuitiveness and non-intrusiveness, stating that it could be easily integrated into their current HIS. However, the authors pointed out some issues related to the usage of RFID tags with sensors for patient monitoring. The RFID reader powers the RFID tags, and when using tags with sensors, the readers need to provide considerably more energy to the tags. The readers must be adjusted to provide enough power, but local regulations limit the transmission power. Regarding e-health standards, the authors did not discuss any protocols for exchanging data such as FHIR, which can undermine the integration of the system with existing HISs.

Wu et al. [15] have developed a system which uses wearable sensor patches to monitor the patients' status. The wearable sensors transmit the different physiological signals (ECG, PPG and body temperature) to gateways using BLE, which can either by fixed (using a Raspberry Pi module) or mobile (using a smartphone app). The gateway exchanges data with the cloud through bridged MQTT brokers, after which it is stored in a MySQL database. The data is stored both in the cloud server and in the fixed gateway. The local users can interact with the system through a web-based user interface (UI) using the smartphone or other web browsers in the local area network. However, the usage of local data storage can cause data integrity issues as the system must ensure databases in both the server and

gateways are synchronized at all times. This can undermine the scalability of the system, as the redundant data synchronization can become a performance bottleneck in the long term.

In [14], the authors proposed a IoT infrastructure that acquires real-time patient data from wearable sensors, using a cloud platform to handle all data processing and storage requirements. The authors have developed a wearable device which takes the form of a sock, designated "CloudSensorSock". The CloudSensorSock acquires mobile data, through accelerometer and gyroscope sensors, vital data, through temperature and heartbeat sensors, and contextual information about the patient's environment using air quality ($CO_2$) sensors. It communicates with a mobile app through BLE, which acts as a gateway to the cloud server. The authors propose moving the data processing entirely to the cloud server as cloud platforms can scale to the needs of the application with little management and cost. However, this approach may not be viable for time critical applications. As discussed earlier, the latency in the communications between devices and remote servers may have a negative impact on the application, especially since the authors propose using this system for detecting fall events.

Recently, and motivated by the COVID-19 pandemic, Raposo et al. [37] developed a system called "e-CoVig" a low-cost solution for monitoring COVID-19 patients during the quarantine.The data acquisition is performed using a mobile app. To collect physiological data, the authors developed a specialized wearable device that communicates with the mobile app through BLE, recording pulse oximetry ($SpO_2$), heart rate, and temperature data. Alternatively, patients can use their own measuring devices, *e.g.* a thermometer, and manually insert the measurements or use Optical Character Recognition (OCR) to automate the in-app insertion of the values. The app can also be used to record audio snippets in order to detect cough and monitor respiratory activity. Unfortunately, the lack of e-health standards hinders its integration with external healthcare systems.

Table 2.5 summarizes the key properties of the previously discussed solutions.

Table 2.5: Comparison between different pervasive healthcare applications.

| **References** | **Measured Signals** | **Networking Protocols** | **Data Storage** | **e-Health Standards** | **Application Features** | **Security Features** |
|---|---|---|---|---|---|---|
| Fuhrer et al. [25] | N/A | EPC/RFID, Wi-Fi | MySQL | None | RTLS | Unspecified Storage Encryption |
| Adame et al. [18] | Temperature, Heart Rate, Accelerometer | EPC/RFID, Wi-Fi | MySQL | None | RTLS, Fall Detection, Vital signs monitoring | AES-128, WPA-Personal |
| Wu et al. [15] | Temperature, Heart Rate, Accelerometer | BLE, Wi-Fi, MQTT | MySQL | None | RTLS, Fall Detection, Vital signs monitoring | AES-128 |
| Doukas et al. [14] | Temperature, Heart Rate, Accelerometer, $CO_2$ Sensor | BLE, Wi-Fi, GPRS/3G HTTP | MySQL | None | Fall Detection, Vital signs monitoring | AES |
| Raposo et al. [37] | Temperature, Heart Rate, Pulse Oximetry, Respiration Rate | BLE, Wi-Fi | Unknown | None | Fall Detection, Vital signs monitoring, Clinical decision support | Unknown |

### 2.3.1 Weaknesses of literature

In the literature, many solutions secure communications between the devices using standard encryption algorithms like Advanced Encryption Standard (AES) [18, 15, 14]. However, very few discuss authentication and authorization processes [14, 23]. To ensure that no data is leaked to malicious authors, the networking protocols used must ensure these **security** properties.

Several web services currently use Transport Layer Security (TLS) [38]. This protocol ensures integrity, confidentiality and authentication as it combines public key cryptography to validate the identity of the communicating parties, symmetric-key algorithms to encrypt the transmissions and message integrity checks to ensure the transmissions are not tampered during transport. These properties make this protocol invaluable for secure communications over the web, and thus should be an integral component of IoT networks. Moreover, access control needs to be considered. Systems are composed by many devices, which may have different levels of access level for each device. For example, limiting access to certain topics in MQTT, so that devices can only subscribe and publish messages in specific topics.

Despite recent efforts, **interoperability** is still an issue for IoT systems. Due to the lack of clear and concise industry standards and regulations, many manufacturers develop their own proprietary data formats and communication protocols, which hampers the integration of new resources since solutions are designed within closed ecosystems [28].

Fortunately, there are several international initiatives to promote the use of IoT in health in a standardized way, such as HIMSS (Healthcare Information and Management Systems Society) and the Personal Connected Health Alliance (PCHAlliance). PCHAlliance, for example, advocates the adoption of the Continua Design Guidelines (CDG), which facilitates the integration of personal health devices into health systems. These guidelines have been recognized by ITU (International Telecommunication Union) and the European Commission and are adopted by countries such as Denmark, Norway and the USA, among others [39]. These guidelines promote a series of e-health standards like FHIR which facilitate the exchange of information between systems, in order to ensure that the implementations become truly interoperable.

## 2.4 Statement of Contributions

In the scope of the WoW R&D project, wearable devices, designated "Biostickers", have been developed to collect physiological patient signals using the BLE networking stack. As an alternative, the project also studied the possibility of using of RFID for the networking stack, in an effort of creating battery-less wearable devices. However, this has shown to be unfeasible due to strict energy consumption and energy transfer requirements that could not be met during development.

Having this in mind, and in the context of this dissertation work, a novel fully modular IoT infrastructure is proposed, making use the FHIR standard in order to fully integrate the data into an existing and widely used HIS – GlobalCare[2] by Glintt - Healthcare Solutions, S.A.

From a hardware perspective, the IoT system is composed of 3 different components, as seen in Figure 2.5:

- the *Biosticker* (and the Oximeter), that acquire the patient's physiological signals;
- the *Smart box*, which acts as a central node of the WBAN and aggregates the data, communicating it to the gateway via Wi-Fi;
- and finally, the *Smart Gateway*, which serves as a fog server in order to mitigate latency and other computing issues and acts as the gateway to the HIS, by communicating with the "Interoperability" FHIR layer on the HIS.

As the goal of the project is the deployment of the system in an hospital, Centro Hospitalar e Universitário de Coimbra (CHUC), each *Smart box* is coupled with a hospital bed, which is referred as a *SmartBed*.

[2]GlobalCare by Glintt: `https://globalcare.glintt.com/`

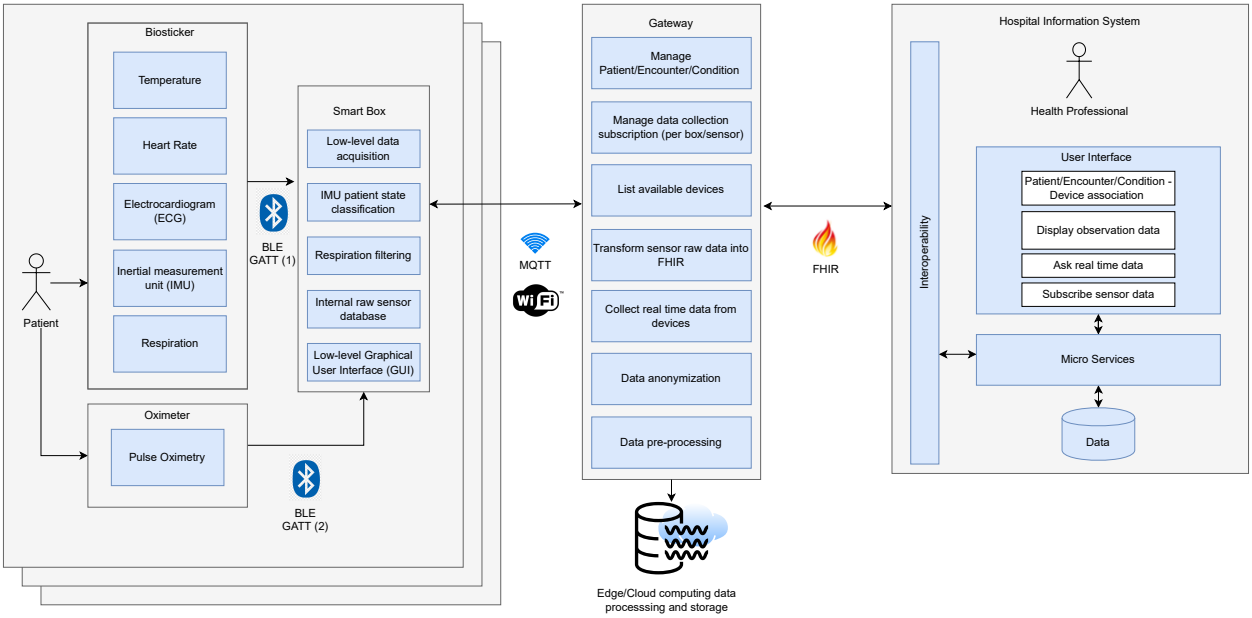


Figure 2.5: System architecture of the WoW project. In this work, the details of implementation of the *Smart Gateway* components are discussed, as well as contributions to the *Smart box* development.

For the work developed throughout the dissertation, the following contributions are proposed:

- Development and deployment of the *Smart boxes* embedded in hospital beds for data acquisition from *Biostickers* attached to patients' skin, namely:
  - Hardware evaluation of 2 different IoT kits (Raspberry Pi and Udoo Bolt).
  - Evaluation of different BLE adapters for data acquisition.
- Design and implementation of data integration pipelines using MQTT and management of the multiple *Smart boxes* in the *Smart Gateway*;
- Usage of a FHIR API layer to integrate the proposed system in the GlobalCare HIS.
- Evaluation of the performance of the proposed system on an hospital trial and through controlled lab tests within the WoW project.

## 2.5 Summary

In this chapter, the importance of IoT systems is discussed, how these systems are composed and how these can bring value to healthcare providers. After analyzing different relevant

systems proposed previously in the literature, a set of criteria was defined to guide the development of our own implementation.

The next chapter begins with the evaluation of the different IoT kits for the *Smart Bed* hardware, and afterwards, an evaluation of different BLE adapters for patient data acquisition to implement secure communications between the *Biostickers* and the *Smart boxes*.

# 3 Smart box Development

In the proposed architecture, the *Smart box* plays the role of acquiring the data that is transmitted wirelessly by the *Biostickers*, which can be seen on Figure 3.1. Each *Smart box* is associated to a single patient, it captures the data of each *Biosticker* attached to that patient, and stores it in a local database for redundancy. It should also be capable of analyzing and processing the data in real time, before propagating it to the higher layers in the system architecture, in order to reduce computation and networking overhead on the Smart Gateway. The WoW project also foresees the usage of a classification algorithm in the *Smart box* to determine the body pose of the patient, as well as filtering the respiration data to account for signal fluctuations caused by sudden movements from the patient.

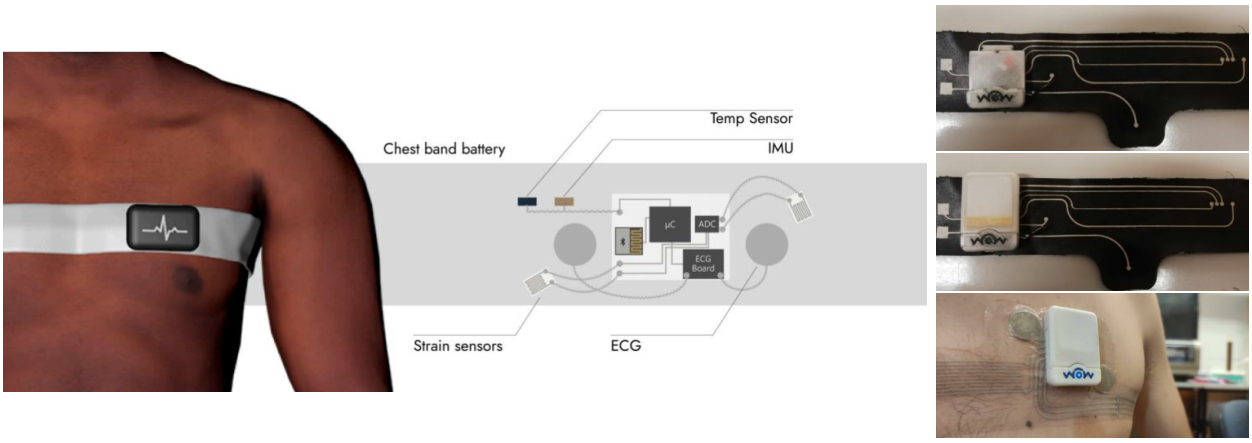


Figure 3.1: Illustration of the *Biostickers*. On the left, a conceptual illustration of the different components that form the *Biosticker* is shown. On the right, photographies of a test volunteer using the *Biosticker* are exhibited.

Additionally, for debugging purposes, researchers at the ISR developed a simple developer UI to be deployed on the *Smart box*, which can be seen on Figure 3.2.

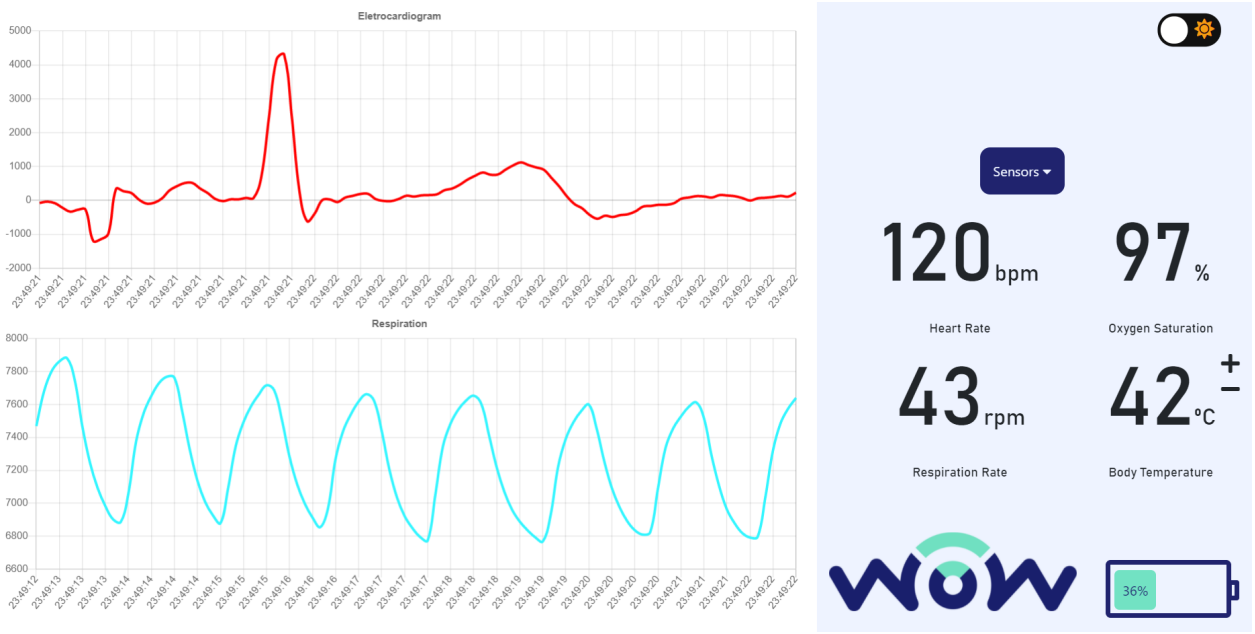


Figure 3.2: Illustration of the developer UI for debugging the *Smart box* acquisition, designed by the WoW research team at ISR.

Regarding data acquisition, the *Smart box* collects 5 distinct biosignals, which can be seen on the developer low level UI on Figure 3.2:

- Electrocardiogram (ECG) – Byte array (20 byte length) with the electrical signal measured, with a frequency of 20 Hz.
- Respiration Rate – An unsigned integer (4 bytes) representation of the rate of respiration, with a frequency of 10 Hz.
- Heart Rate – An unsigned byte representation of the heart rate in *beats per minute*, every 5s.
- Body Temperature – An IEEE 11073[3] floating-point number representation of the body temperature, every 60s.
- Oxygen Saturation – A standard floating-point number (4 bytes) representation of the oxygen saturation, every 1s.

[3] `https://standards.ieee.org/standard/11073-10207-2017.html`

## 3.1 Deciding on a Hardware Platform

As discussed, there are multiple requirements the *Smart box* hardware must fulfill, from communication to the *Biostickers* to integration with the *Smart Gateway*:

- The device must be small enough to be mounted on a hospital bed.
- The device must be powered via a standard electrical outlet, so power consumption constraints are not an issue.
- It must support BLE 5.0, and capable of handling the communication bandwidth required by the *Biostickers*.
- The hardware must support Ubuntu 20.04 LTS as its operative system.
- It must communicate to the *Smart Gateway* using the protocol MQTT, connected via Wi-Fi.

With these requirements in mind, and in context of this dissertation, two different Single Board Computers (SBC) have been considered for the development of the *Smart box*: a Raspberry Pi 4 Model B and an UDOO BOLT v3. In the following sections the characteristics of each platform are discussed and compared.

### Raspberry Pi 4 Model B

Raspberry Pi denotes a series of SBCs which are developed by the Raspberry Pi Foundation, a UK-based charity that aims to educate the general public about the power of computing and digital making, in association with Broadcom. It is one of the most popular hardware platforms used by developers due to its accessible price and community support [40]. At the time of the writing, the Raspberry Pi 4 Model B (or Raspberry Pi 4B)[4], which can be seen in Figure 3.3, is the latest revision of the Raspberry Pi series, powered by Broadcom BCM2711 System on a Chip (SoC).

[4] `https://www.raspberrypi.com/products/raspberry-pi-4-model-b/`

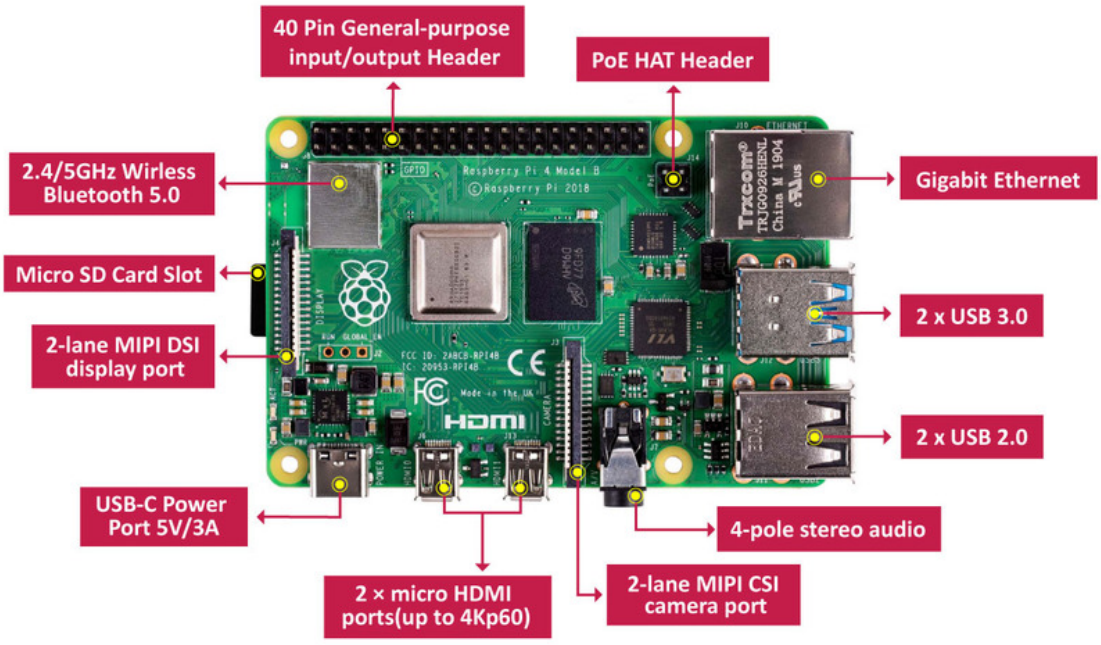


Figure 3.3: Raspberry Pi 4B.

### UDOO BOLT V3

As stated by the manufacturer[5], the "UDOO BOLT is a quantum leap compared to current maker boards". It represents a series of high performance SBC, equipped with the latest generation of AMD Ryzen Embedded SoC. Additionally, it contains an Arduino Compatible microcontroller (connected via UART), making the UDOO BOLT extremely versatile. UDOO BOLT is incredibly well-supported by UDOO but unfortunately, it still does not have nearly the same community support of Raspberry Pi.

The UDOO BOLT V3, which can be seen in Figure 3.4, is the entry-level product of the series, but it is still capable of allegedly outperforming full-fledged computers such as the Apple MacBook Pro 13", which just goes to show how powerful these SBCs can be.

[5] `https://www.udoo.org/discover-the-udoo-bolt/`

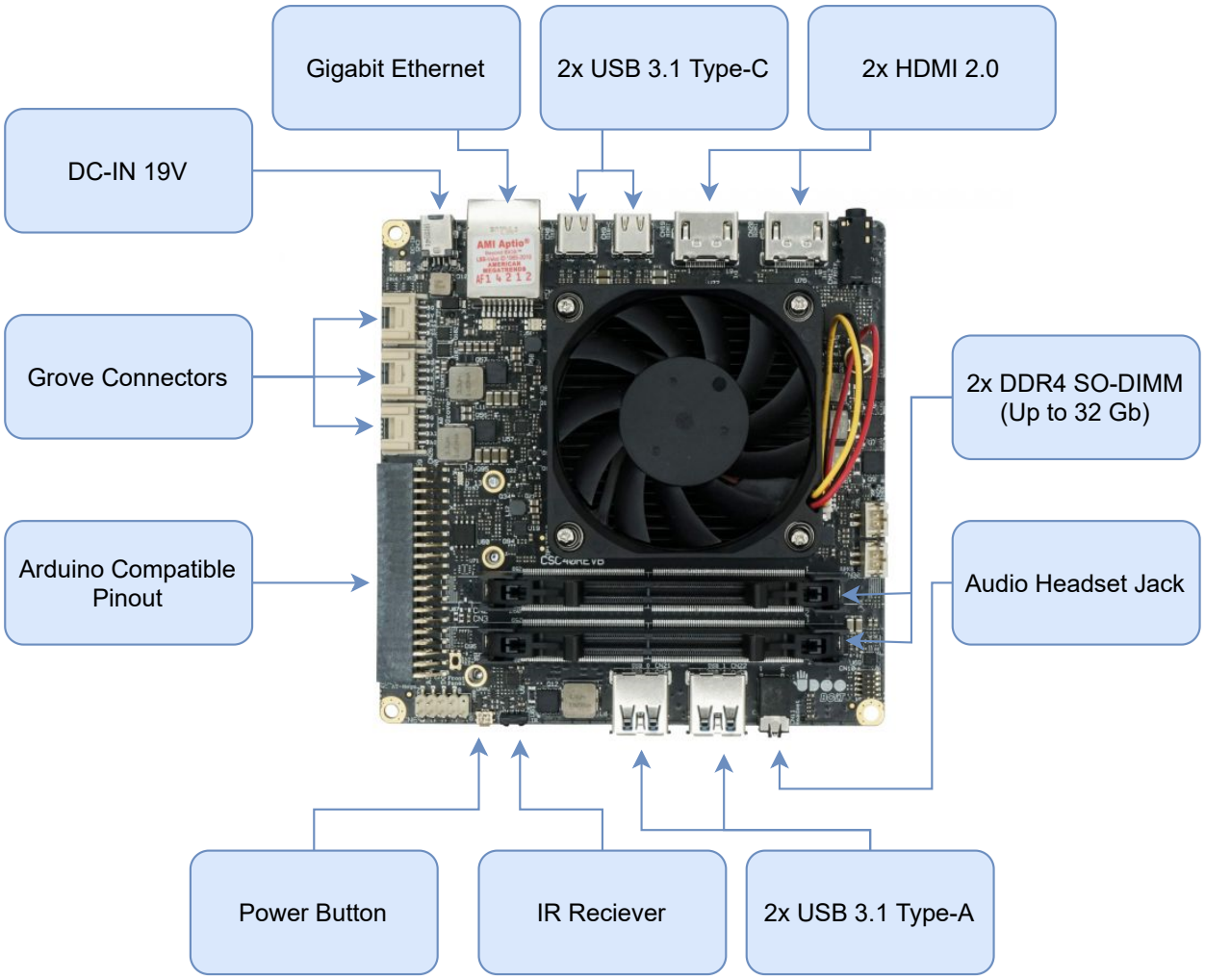


Figure 3.4: UDOO BOLT V3.

### 3.1.1 Comparing the Hardware Platforms

In order to decide on which platform to pick for the development of the project, it is crucial to compare the specification of both boards, which can be seen in the Table 3.1.

From Table 3.1, it can be concluded that the Raspberry Pi is a much more affordable alternative. At over 1/7 of the price, it already has a working Wi-Fi+BT networking module (which is not included in the UDOO BOLT V3), nearly identical Input/Output (I/O) port capability and a smaller size. UDOO BOLT V3 on the other hand, has a much better SoC, which is expected to deliver a much better overall computing performance.

Table 3.1: Specifications of the Raspberry Pi 4B and UDOO BOLT V3.

| | **Raspberry Pi 4B** | **UDOO BOLT V3** |
|---|---|---|
| **SoC** | Broadcom BCM2711<br>(ARMv8 64-bit)<br>4-core @ 1.5G Hz | AMD Ryzen™ Embedded V1202B<br>(AMD64 64-bit) 2-core @ 2.3G Hz<br>(up to 3.2G Hz turbo) |
| **RAM** | 2, 4 or 8 GB LPDDR4 | Up to 32 GB DDR4 (Not included) |
| **Storage** | No internal storage,<br>SDXC Card Support | 32 GB internal eMMC +<br>1 × SATA III and<br>2 × M.2 connectors |
| **Networking** | 2.4/5.0 GHz Wi-Fi, Gigabit<br>Ethernet, Bluetooth 5.0, BLE | Gigabit Ethernet + M.2 Key E slot<br>for optional Wi-Fi+BT module |
| **I/O Ports** | 2 × USB 3.0, 2 × USB 2.0,<br>2 × (Mini) HDMI | 2 × USB 3.0 Type-A,<br>2 × USB Type-C (w/ Display Port<br>+ Power Delivery), 2 × HDMI |
| **Other Features** | Power over Ethernet<br>(PoE)–enabled | Includes ATmega32U4 microcontroller<br>(Arduino Leonardo compatible),<br>RTC Battery |
| **Dimensions** | 8.5 x 5.6 x 1.7 cm | 12 x 12 x 7 cm |
| **Max. Power Consumption** | 9 W | 60 W |
| **Price** | 75.93 € (**8 GB Model**,<br>including a 32 GB SDXC Card<br>and case) | 534.48 € (including external power<br>supply and a 16 GB RAM module) |

In order to understand how these differences in the hardware specification between the SBCs translate to real-world performance, a test suite has been developed and conducted to quantify the performance of each SBC. The tools developed for each test can be found here[6]. In the next sections, the details of each test are explained and the performance of each SBC is discussed.

### Test 1: Python Benchmark

Given the data processing requirements for the *Smart box*, and as Python is used as the main scripting language for most of the *Smart box* development, a simple test has been developed to estimate computing performance, or more specifically (single-threaded) performance of arithmetic tasks, on each SBC. In this test, each SBC calculates the $n$-th number in the Fibonacci sequence [41], and the time taken to complete is measured. This process is repeated 10 times for different numbers, from 10000 to 500000, to determine average run time.

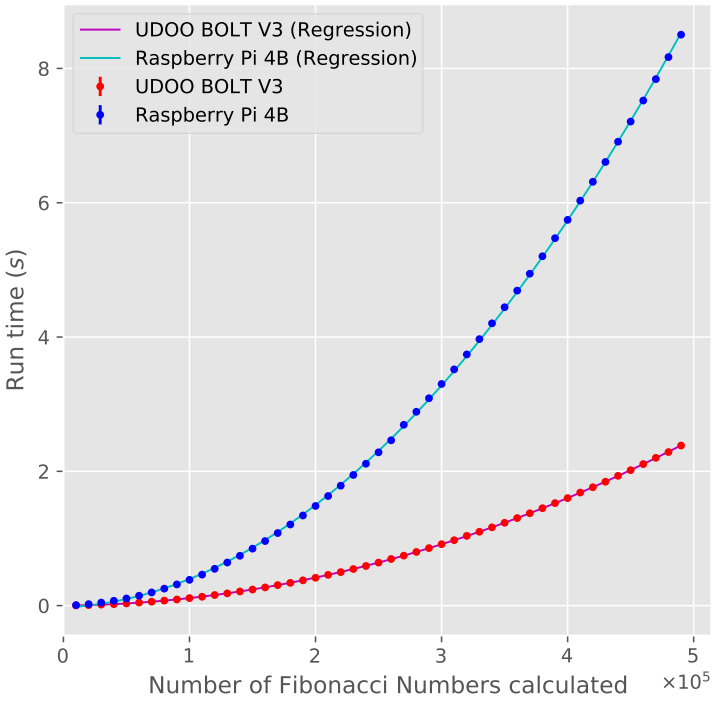


Figure 3.5: Custom Python benchmark for the Raspberry Pi 4B and UDOO BOLT V3.

From Figure 3.5, it can be observed that the time taken for computing each number increases quadratically for each platform ($R^2 > 0.99$ for both regressions), as expected, with

[6] `https://github.com/WoW-Institute-of-Systems-and-Robotics/smartbox_benchmark_tests`

UDOO BOLT V3 outperforming Raspberry Pi 4B on average by a factor of 2.5 ± 0.2.

**Test 2: Phoronix Test Suite**

The Phoronix Test Suite[7] is an open-source benchmarking platform used for comparing the performance of different systems. The framework provides compilations of tests for a variety of tools and is also fully customizable and expandable, allowing users to develop and automate their own tests in a clean, reproducible and easy-to-use fashion. The test profiles work by measuring some property of the benchmark, (*e.g.* the run time for calculating the first 100 Fibonacci numbers) and use it to provide an estimate of the performance of the SBC, which can be easily used for comparison between different systems.

For the purposes of evaluating the computing performance of each SBC, the following standard test profiles provided by Phoronix[8] were chosen, which are a compilation of the most popular Python and Central Processing Unit (CPU) benchmarks used[9]:

- BYTE Unix Benchmark ("BYTE"), single-threaded CPU benchmark – Runs BYTE UNIX benchmark suite (more accurately, the Dhrystone 2 synthetic benchmark) to measure the amount of instructions per second (IPS).
- 7-Zip Compression ("7-Zip"), multithreaded CPU benchmark – Runs the benchmark feature integrated in 7-Zip to measure the amount of millions instructions per second (MIPS). The benchmark consists of a LZMA data compression and decompression test run, using all available threads in the system (meaning it will scale highly with the amount of threads in the system).
- PyBench Benchmark ("PyBench"), single-threaded Python & CPU benchmark – Executes different function such as built-in function calls and nested for-loops and measures its runtime.
- PyPerformance *chaos* Benchmark ("chaos"), single-threaded Python & CPU benchmark – Create chaos game-like fractals [42] and measures its run-time.

[7] `https://www.phoronix-test-suite.com/`

[8] `https://openbenchmarking.org/`

[9] The benchmark results obtained were published in the *OpenBenchmarking.org* website, and can be found in the following URL: `https://openbenchmarking.org/result/2110255-JNCF-211025851`.

- PyPerformance *float* Benchmark ("float"), single-threaded Python & CPU benchmark – Create 100,000 random floating-point numbers and calculate the co-sine, sine and square root of each one and measures its run-time.

- PyPerformance *nbody* Benchmark ("nbody"), single-threaded Python & CPU benchmark – Runs an *n-body* problem simulation [43] and measures its run-time.

- PyPerformance *json_loads* Benchmark ("json"), single-threaded Python & CPU benchmark – Evaluates JavaScript Object Notation (JSON)[10] parsing and serialization, a widely used open standard data format, by dumping and loading thousands of objects and measures its runtime.

- PyPerformance *crypto_pyaes* Benchmark ("crypto"), single-threaded Python & CPU benchmark – Runs the AES block-cipher Python implementation and measures its run-time.

- PyPerformance *regex_compile* Benchmark ("regex"), single-threaded Python & CPU benchmark – Compiles different *regular expressions* or *regexes* in Python and measures its run-time.

- PyPerformance *python_startup* Benchmark ("startup"), single-threaded Python & general system performance benchmark – Measures Python's startup time.

- PyPerformance *django_template* Benchmark ("django"), single-threaded Python benchmark – Builds a 150x150-cell HTML table and measures its run-time.

[10]The JavaScript Object Notation (JSON) Data Interchange Format: `https://www.rfc-editor.org/rfc/rfc8259.html`

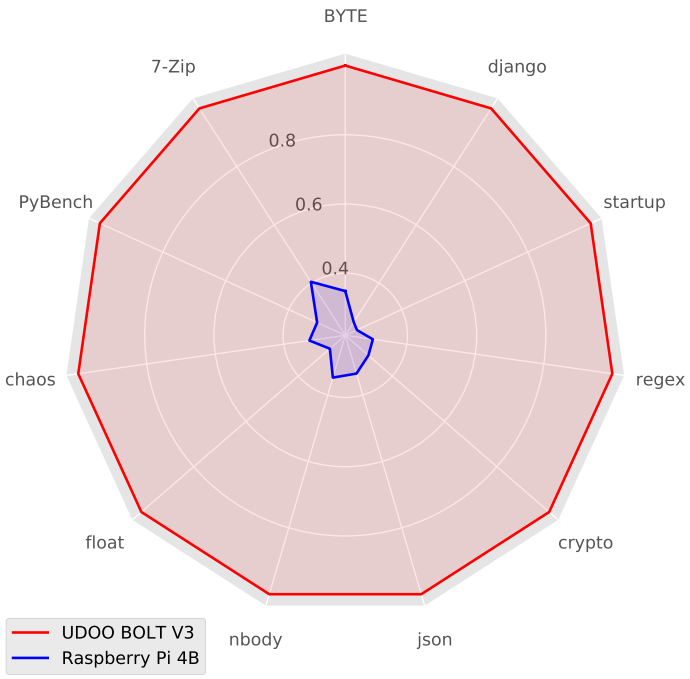


Figure 3.6: Phoronix benchmarks for the UDOO BOLT V3 and Raspberry Pi 4B. The performance values for each test are normalized to UDOO BOLT V3 performance.

From Figure 3.6, it can be observed that UDOO BOLT V3 performs much better than Raspberry Pi 4B in every benchmark, as expected. However, some disparity in the relative performance throughout the benchmarks is also noticed.

For example, in the 7-Zip benchmark, this is likely due to the fact that is a multithreaded test, and as Raspberry Pi 4B has more threads than UDOO BOLT V3, it manages to close the performance gap (albeit only slightly). For the *startup* and *django* benchmark score differences, these are mostly affected by the load time of the Python interpreter, which is significantly affected by the main storage device. A high-speed SD card was used with the Raspberry Pi and the internal eMMC storage was used with the UDOO BOLT V3, which is a significantly faster storage medium than an SD card and thus is most likely the cause for the benchmark score difference. Benchmarks which use many arithmetic operations, like the *float* benchmark or *BYTE*, can also show significantly different results as well since each CPU architecture has different optimizations for handling floating point operations or integers operations.

On average, UDOO BOLT V3 outperforms Raspberry Pi 4B in all benchmarks by a factor of $2.21 \pm 0.4$.

### Test 3: MQTT Benchmark

As the *Smart box* is intended to communicate with the *Smart Gateway* through MQTT messages, an evaluation of how each system handles the load associated with a MQTT client is performed. For this test, each SBC uses a simple MQTT client, that subscribes to a single topic and publishes a message that same topic for different transmission rates, with a payload with a length of 1000 bytes, through a MQTT broker in the local network. Additionally, the tests measure the communication with and without TLS to evaluate the impact of the secure protocol.

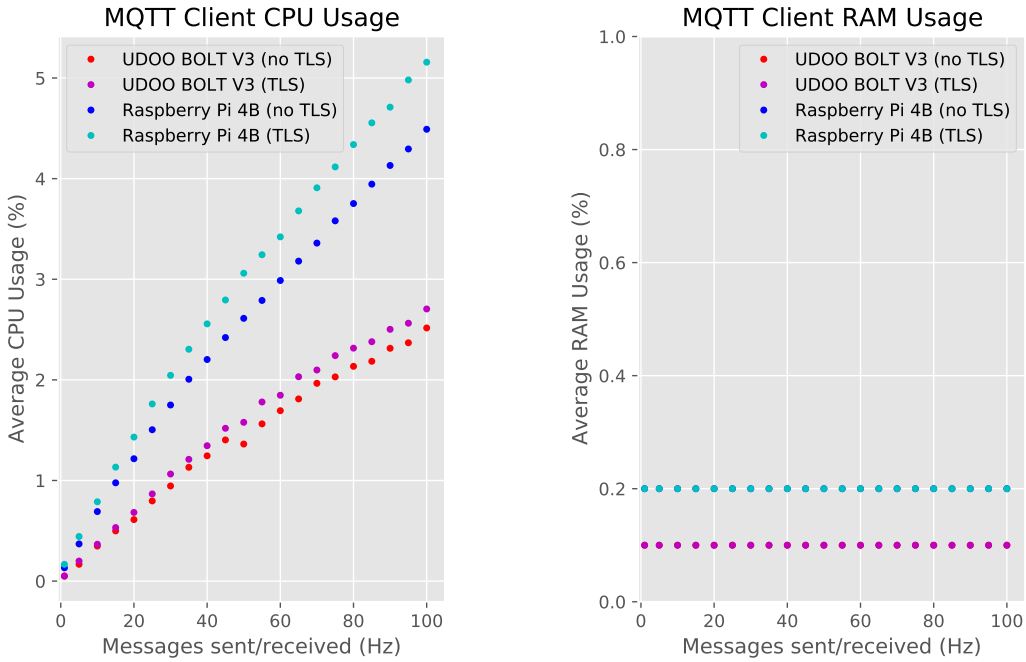


Figure 3.7: MQTT benchmark for Raspberry Pi 4B and UDOO BOLT V3. In the RAM usage graph, the tests with TLS and no TLS show identical results, causing the data points to overlap.

The MQTT client is a very lightweight process, and as seen in Figure 3.7, it is capable of running on both platforms with trivial performance impact. As the transmission frequency increases, the CPU load increases linearly, Additionally, it can be observed that the Raspberry Pi uses more CPU than the UDOO BOLT V3, by a factor of $1.88 \pm 0.21$ without using TLS and $2.01 \pm 0.31$ using TLS. The usage of RAM remains constant throughout the tests, 0.1% on UDOO BOLT V3 and 0.2% on Raspberry Pi 4B – which are very negligible values, and thus do not represent any significant impact in performance. Additionally, the usage

of TLS directly impacts the performance of the devices, but not to any significant degree – $0.16\% \pm 0.02$ more CPU usage on the Raspberry Pi and $0.094\% \pm 0.039$ on UDOO BOLT V3.

Overall, the UDOO BOLT V3 shows a much better results, with nearly twice less CPU and RAM usage. However, even at transmission and reception rate of 100 Hz, the resource usage of both devices is very minimal (<6%) and thus should not meaningfully impact real-world performance.

### 3.1.2 Final Decision on Smart box hardware

Based on the results of our tests it is concluded that the UDOO BOLT V3 heavily outperforms the Raspberry Pi 4B in CPU benchmarks, but shows a negligible difference in memory usage. Nonetheless, these performance gains do not meaningfully impact the *Smart box* functionality, for example, in the MQTT communication. This observation can also be extended to other functionalities, such as the developer UI or the Python acquisition script, which should have a similar resource load to the MQTT client in the MQTT benchmark, so both SBCs are more than capable of assuring the necessary functions.

Additionally, the Raspberry Pi 4B comes out of the box with an included Wi-Fi+BLE combo networking card, 8 GB of RAM (as the 8 GB model was chosen), and a much smaller form factor – which is very useful for embedding the *Smart box* in the *SmartBeds*. And this is at 1/7 the cost of UDOO BOLT V3, making this much more affordable to scale and replicate the system with many *Smart boxes*.

Due to all the aforementioned reasons, Raspberry Pi 4B was chosen for the *Smart box* development in the scope of the WoW project.

## 3.2 Communication with the Biostickers

As previously mentioned, the communication between the *Biostickers* and the *Smart box* makes use of the BLE protocol. One of the advantages of using the Raspberry Pi 4B, as discussed in the previous section, is the fact that it already includes all networking functionality needed for the project.

However, the communication with the *Biostickers* is very critical and demanding. During the development of the project, the research team has decided to use a *Biosticker* coupled with a commercially available Pulse Oximeter– used to measure oxygen saturation at the fingertip – in order to capture all required biosignals. This means that the BLE acquisition system must be capable of handling both communications simultaneously. With this in mind, it is necessary to verify if the included BLE adapter of the Raspberry Pi board is sufficient for the task, or if a different acquisition hardware should be considered instead.

In order to understand how data transmission works between BLE devices, some technical background regarding the data transmission in protocol is presented.

### 3.2.1 Technical Background

The BLE protocol stack is organized into three major components, as shown in Figure 3.8: the Application Layer, the Host Layer and Controller Layer.

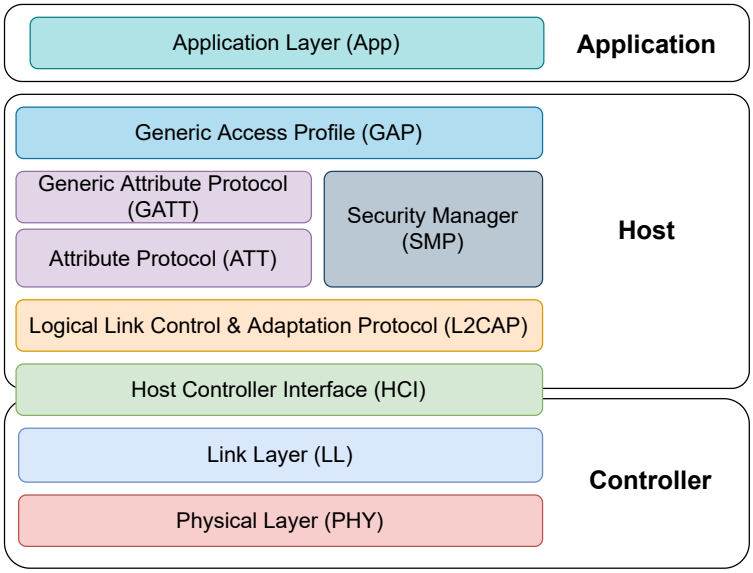


Figure 3.8: Diagram of the different components of the BLE protocol stack. Adapted from [44, 45]

The Physical Layer (PHY) and Link Layer (LL) components constitute the Controller layer. The PHY is the bottom layer of the BLE stack, and is responsible for the transmission and reception of information over radio waves on the Industrial Scientific Medical 2.4GHz band. According to the latest revision of the specification [44], BLE supports 3 distinct physical layers: LE 1M, LE 2M and LE Coded. Each of these define which modulation speed (1 Msym/s or 2 Msym/s) and which coding scheme (S=1,2 or 8 bits) is used. LE 1M

is the default PHY which must be supported by BLE devices, with a bit rate of 1 Mbps (1Msym/s with S=1). In the latest major revision of Bluetooth, the latter two PHYs were introduced: LE 2M doubles the bit rate to 2 Mbps using a faster modulation scheme (2 Msym/s with S=1), whereas LE Coded has a much larger range (2x or 4x compared to LE 1M), at cost of a lower bit rate (500 kbps or 125 kbps) by using a different coding scheme (1 Msym/s with S=2 or 8 bits respectively).

The Link Layer (LL) interfaces directly with the PHY, and manages the link state of the radio. It also provides the mechanism for the higher layers to interact with the radio transceiver.

Following the Link Layer (LL) is the Host Layer, containing the higher level components of the protocol stack that interact with the application level layers. The L2CAP is responsible for managing the data between the LL through the Host Controller Interface (HCI) and the higher layers in the protocol stack. It abstracts the communication details from the higher layers, handling seamlessly the fragmentation of the data into multiple LL data packets for transmission and reassembly of LL data packets for higher layer protocols, such as the Attribute Protocol (ATT).

ATT is the protocol used to expose the application data to other BLE devices through data structures called "attributes", which are the smallest addressable units of data used by ATT. These entail three main properties [44]:

- An attribute type, defined by a Universally Unique Identifier (UUID)[11], which indicates the type of data that is stored in the attribute.
- An attribute handle, to uniquely identify that attribute on the device (which in this case assumes the role of a server), allowing other devices (which assume the role of clients) to refer the attribute for read and write requests;
- A set of permissions which limit the types of interactions.

However, from the application point of view, data is exchanged using Generic Attribute Profile (GATT). GATT defines a service framework using the ATT protocol, with the procedures used to exchange data between the BLE devices. Regarding GATT, there are three main constructs to consider:

[11] `https://tools.ietf.org/html/rfc4122`

- GATT Characteristic – The lowest level concept in GATT transactions is the Characteristic, which encapsulates a single data point, *e.g.* a temperature measurement, an accelerometer reading, etc. It also defines the different types of interactions, such as reading, writing, subscribing for notifications, etc.
- GATT Service – A collection of GATT characteristics.
- GATT Profile – A collection of Services, usually predefined by Bluetooth Special Interest Group (SIG) in order to promote interoperability, which may also be customized for the application's needs.

The Security Manager (SMP) handles security in data transmissions, containing the security algorithms used to encrypt and decrypt the communication.

Finally, the Application Layer contains the user application, which interfaces with the Bluetooth protocol stack.

**BLE data transmission**

Before discussing BLE data transmissions, it is important to introduce to certain terminology which is commonly used:

- Central device (or master): Device that initiates commands and requests.
- Peripheral device (or slave): Device that receives commands and requests, and returns responses.
- Connection Event: Moment of the connection where the devices engage in radio transmissions.

As seen previously, there are multiple layers that are involved in the transmission of data in BLE. Figure 3.9 displays the format of a BLE data packet, showing how the data is encapsulated for the transmission.

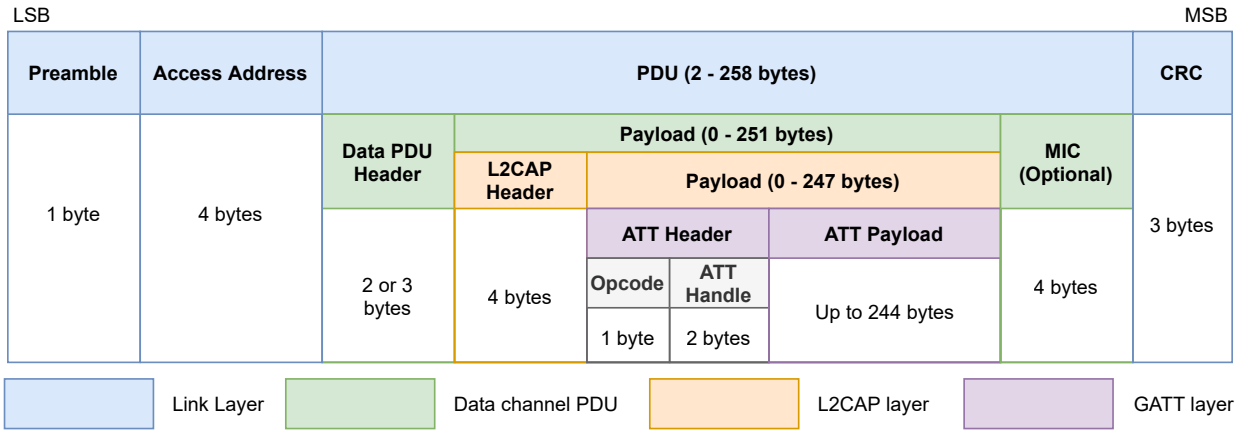


Figure 3.9: BLE data packet format for an ATT write message using LE 1M PHY. Adapted from [44, 45]. The Message Integrity Check (MIC) and Cyclic Redundancy Check (CRC), which are not discussed above, are used for validating the integrity of the data packet.

Communication in BLE works in a particular manner. Instead of having the devices transmit data whenever there is new information to exchange, the devices group their radio transmissions in a brief time frame that occurs periodically, called the Connection Event [44]. This allows devices to heavily reduce power consumption, as they can negotiate the amount of radio "downtime" to minimize the amount of transmissions.

Figure 3.10 illustrates a Connection Event during a BLE communication between two devices exchanging a single message, A and B. When the device A wants to communicate to the device B, the data must be first fragmented into different LL data packets on the L2CAP layer, which for this simple example is not considered. The data is transmitted to device B through the LL layer, and the data is then reassembled on device B. When the packet is received on device B, it must send a transmission to device A acknowledging the reception of the packet (which contains an empty payload), as device A can only continue transmitting after receiving the acknowledgement [44].

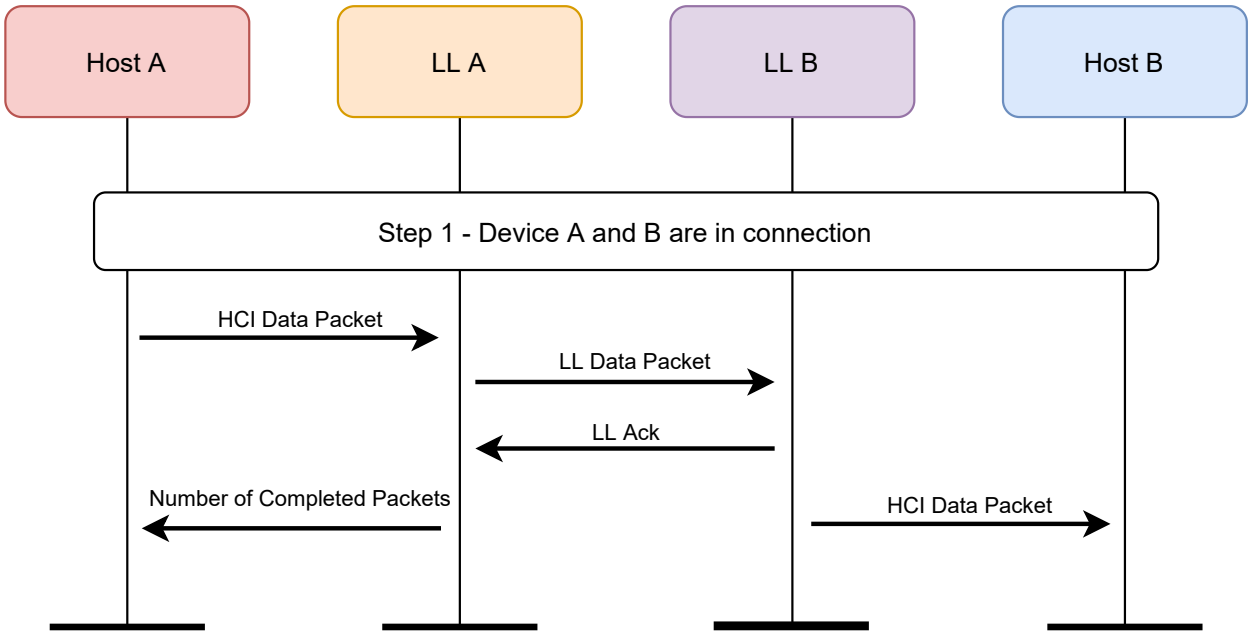


Figure 3.10: Message sequence chart between two BLE devices during a Connection Event. Adapted from [44].

Multiple parameters guide how BLE data connections are performed [44], which are the following:

- Slave Latency: Number of consecutive connection events that the slave device is not required to listen for the master. It allows a slave to use a reduced number of connection events, thus minimizing power consumption.
- Connection Interval: Time between two consecutive connection events.
- Supervisor Timeout: Maximum time between two received data Protocol Data Units (PDU) before the connection is considered lost.
- Maximum Transmission Unit (MTU) for the ATT protocol: Length of the PDU for the ATT protocol, which includes the ATT header as well as the payload containing the data to be transmitted.
- Connection PHY: Modulation and coding scheme used for the connection.

### BLE protocol stack on Linux

In the WoW project, the data acquisition has been developed using the official Linux implementation of the BLE protocol stack [46] – *BlueZ* – in order to promote interoperability; since these drivers are developed and used by the community, allowing us to use multiple

BLE adapters, and even different SBCs, using the same codebase without being tied down to proprietary code.

### 3.2.2 Choosing a BLE adapter

As mentioned previously, one of the objectives of this dissertation work is to analyze if the internal BLE adapter provided by the Raspberry Pi 4B is sufficient for the WoW project. To achieve this, the adapter is compared with another commercially available adapter that met the requirements for the project. These are the following:

- The adapter must support, at least, the Bluetooth 5.0 core specification.
- The adapter must natively support Ubuntu 20.04 LTS, as well as the *BlueZ* BLE protocol stack.

After investigating the available market, the ASUS USB-BT500 USB adapter[12] was chosen due to its affordability and availability, making it an adequate fit for the project's needs.

In the next section, multiple tests are conducted to evaluate the performance of ASUS USB-BT500 against the internal BLE adapter in the Raspberry Pi 4B to reach a decision on which BLE adapter hardware should be used for the transmission of data from the *Biosticker* to the *Smart box*.

### 3.2.3 Testing BLE Communication

To ensure that a BLE adapter is capable of handling the communication on the *Smart box*, two different tests were designed to evaluate their performance at different distances:

*a)* A test to evaluate the roundtrip time for a single message (for different sizes).

*b)* A test to evaluate the maximum bandwidth achievable at a given distance.

For these tests, it should be ensured that the conditions are very similar to those when acquiring data from the *Biostickers*. To this end, the same microcontroller is used – nRF52832 SoC – which is used in the *Biostickers*, with our own custom firmware for the tests. Therefore, the device which is used for the tests is the nRF52-DK developer kit[13]. The firmware

[12] `https://www.asus.com/Networking-IoT-Servers/Adapters/All-series/USB-BT500/`

[13] `https://www.nordicsemi.com/Products/Development-hardware/nrf52-dk`

for the microcontroller is built using the latest version of MbedOS[14], an operative system for ARM-based microcontrollers, and can be found here[15].

### Test Conditions

During the preparation and development of these tests, it was discovered that the ASUS USB-BT500 supported an optional functionality which was not documented in its specification – Data Length Extension (DLE). This feature, introduced in Bluetooth 4.2, allows LL data packet payloads to increase significantly in size, up to 251 bytes (compared to the 23 bytes when not using this feature). It is also worth noting out of the 251 bytes, only 244 bytes can be used after taking into account the L2CAP and ATT overhead in the data packet for write requests, as seen in Figure 3.9.

Additionally, LE 2M or LE Coded could not be used for the tests as the BLE stack seemingly did not support these features, despite claiming support for them and all devices used supporting these features.

In order to ensure replicability and reliability of the test results, the same connection parameters are used for all tests, which are the following:

Table 3.2: BLE connection parameters used for the ASUS USB-BT500 adapter.

| | |
|---|---|
| **Connection Interval** | 7.5 ms |
| **Slave Latency** | 0 |
| **Supervisor Timeout** | 500 ms |
| **ATT MTU Length** | 247 bytes |
| **PHY** | LE 1M |

Table 3.3: BLE connection parameters used for internal Raspberry Pi 4B adapter.

| | |
|---|---|
| **Connection Interval** | 7.5 ms |
| **Slave Latency** | 0 |
| **Supervisor Timeout** | 500 ms |
| **ATT MTU Length** | 23 bytes |
| **PHY** | LE 1M |

[14] `https://os.mbed.com/mbed-os/`

[15] `https://github.com/WoW-Institute-of-Systems-and-Robotics/ble-test-firmwares/`

These tests were conducted indoors, with the devices (the BLE adapter and the microcontroller) in clear view of each other. All tests were performed at different distances: 0, 3, 6 and 9 meters. For distances greater than 9 meters the communication was observed to be too unstable for both adapters. Figure 3.11 ilustrates the setup used for all BLE tests.

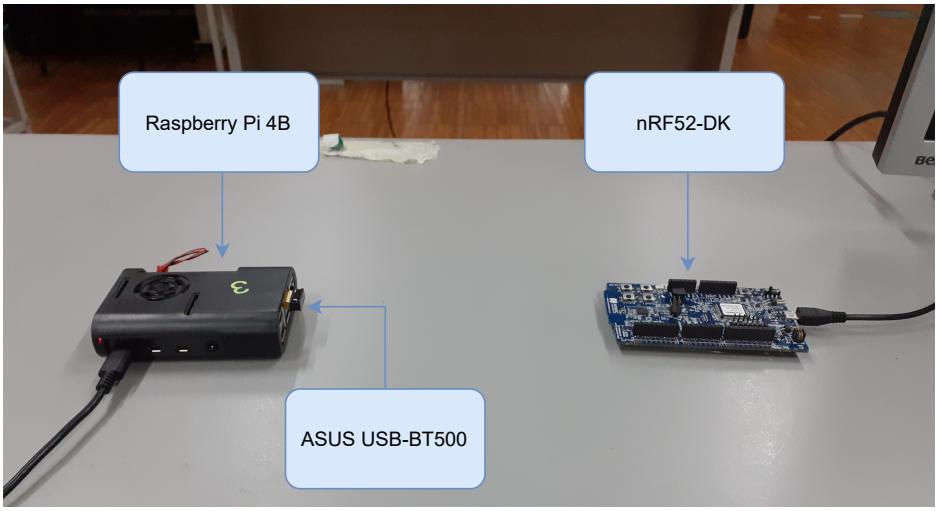


Figure 3.11: Setup used for all BLE tests.

### Test 1: Roundtrip Time Measurement

In this first test, the roundtrip time of a single ATT data packet on a BLE connection is measured for different payload sizes. For this test, the nRF52-DK board exposes a custom GATT service containing 2 characteristics: one is updated by the *Smart box* (characteristic "A"), and one to which the *Smart box* subscribes for notifications (characteristic "B"). When the *Smart box* writes on characteristic "A" on the nRF52-DK GATT server, the GATT server changes the value of the characteristic "B", which triggers the transmission of a notification to the *Smart box*. Thus, the roundtrip time measured is the time taken between the write request on characteristic "A" and the reception of the notification of characteristic "B".

The roundtrip time is measured for different ATT payload lengths, from 1 byte to the maximum size supported by the adapter in 2 byte increments – *i.e.* payload size $ps = \{1, 3, 5, \ldots, 19\}$ bytes for the internal Raspberry Pi 4B and $ps = \{1, 3, 5, \ldots, 243\}$ for ASUS USB-BT500, and for different distances $d = 0, 3, 6, 9$ m. Each test configuration (*i.e.* payload size and distance) has been run 5 times to ensure the consistency of the results, for a total of 2640 independent tests. Figures 3.12-3.19 show the average values measured and their standard deviations for different distances.

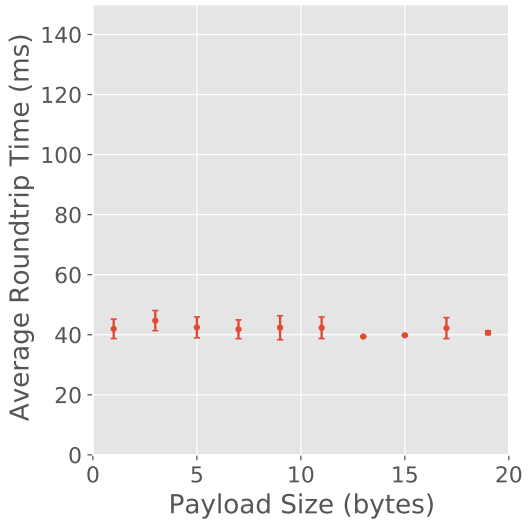


Figure 3.12: Average BLE connection roundtrip time obtained using Raspberry Pi 4B's internal BLE adapter at a distance of 0 m.

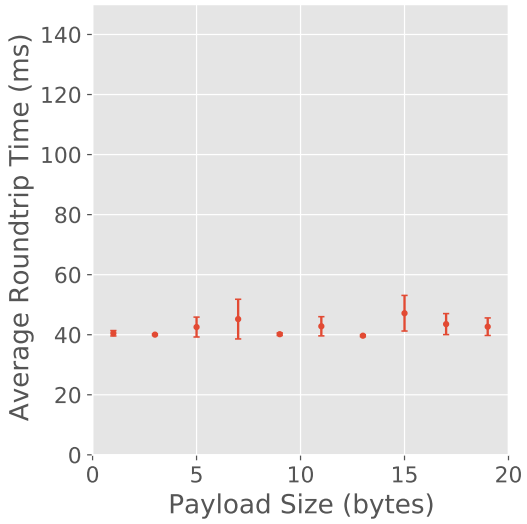


Figure 3.13: Average BLE connection roundtrip time obtained using Raspberry Pi 4B's internal BLE adapter at a distance of 3 m.

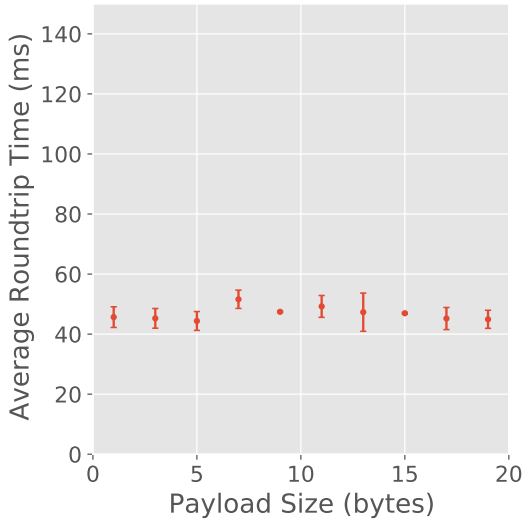


Figure 3.14: Average BLE connection roundtrip time obtained using Raspberry Pi 4B's internal BLE adapter at a distance of 6 m.

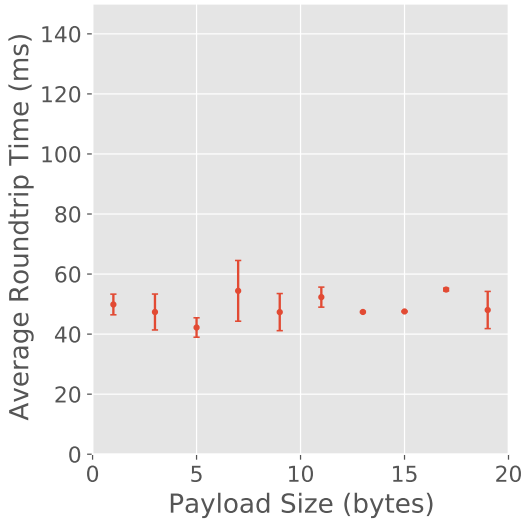


Figure 3.15: Average BLE connection roundtrip time obtained using Raspberry Pi 4B's internal BLE adapter at a distance of 9 m.

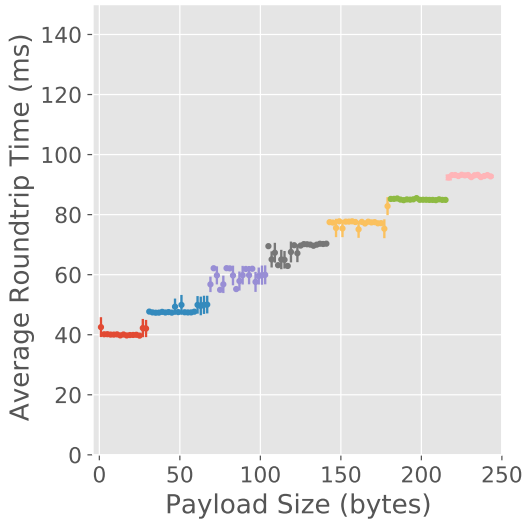


Figure 3.16: Average BLE connection roundtrip time obtained using the ASUS USB-BT500 adapter at a distance of 0 m. The data takes the shape similar to that of a "stepping function", as such, different colors are used to highlight clusters of data that correspond to different steps.

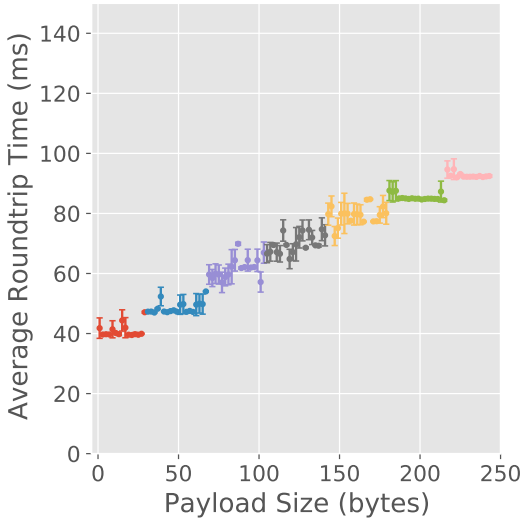


Figure 3.17: Average BLE connection roundtrip time obtained using the ASUS USB-BT500 adapter at a distance of 3 m. The data is colored according to the clusters of data found in Figure 3.16.

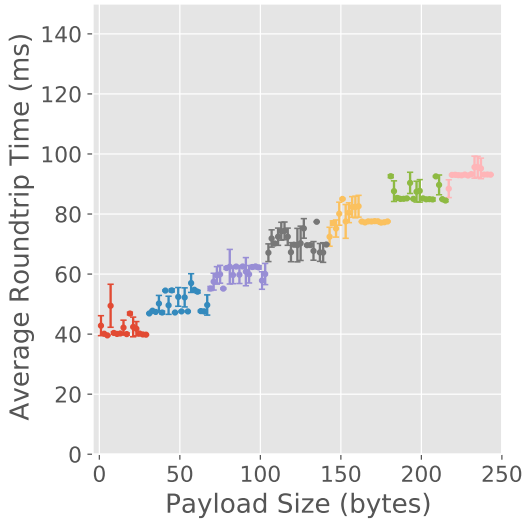


Figure 3.18: Average BLE connection roundtrip time obtained using the ASUS USB-BT500 adapter at a distance of 6 m. The data is colored according to the clusters of data found in Figure 3.16.

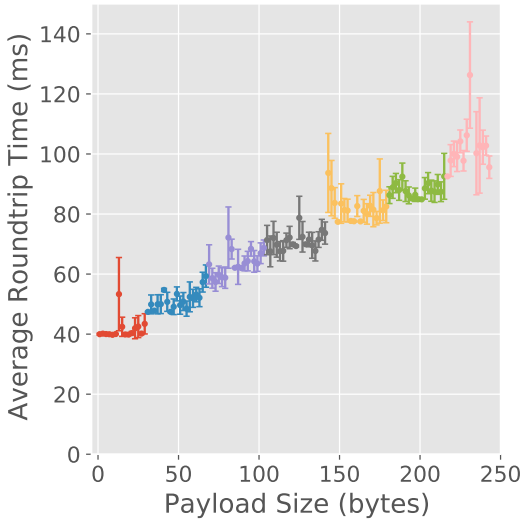


Figure 3.19: Average BLE connection roundtrip time obtained using the ASUS USB-BT500 adapter at a distance of 9 m. The data is colored according to the clusters of data found in Figure 3.16.

Theoretically, the roundtrip time for different payload sizes in a BLE connection should be a single line, or at most a step function as seen in Figure 3.19. This occurs because the amount of radio time allocated for each connection event is limited according to the connection parameters that are exchanged by both devices. If the device is not capable of sending the entire message in the radio time allocated for a single PDU, then it is fragmented over multiple PDUs which can span over multiple Connection Events. Since these occur periodically according to the Connection Interval exchanged at the start of the connection, it creates the "step" effect mentioned. Additionally, even though the transmission time to exchange the payload increases linearly with the size of the payload, from the Application Layer standpoint, this may not be directly observed since most BLE stacks handle the communication differently. All BLE stacks used in these tests notify the Application Layer only after processing all radio events, therefore it should not be possible to receive a message and reply to it in the same Connection Event, thus the transmission of any message must take at least one Connection Interval to be received and processed by the device.

In all tests, both adapters were observed to have a minimum roundtrip time: approximately 40 ms. This is a somewhat unexpected value, given that the connection interval is set to 7.5 ms. This means that the time it takes to transmit a single message and receive it on the *Smart box* takes more than 4 connection intervals, which is more than double of what is expected: one interval for the transmission of the packet to the nRF52-DK and one interval to retrieve the packet, totaling $7.5 + 7.5 = 15$ ms + processing time overhead at the application level. Since the roundtrip time accounts for both the transmission and reception of the data packet, this indicates that a $\sim$12 ms delay exists in every transmission and reception of data.

As discussed in Section 3.2.3, the DLE functionality support gives a head start for the ASUS USB-BT500, as it is capable of sending over 10 times more data in a single packet, compared to the Raspberry Pi 4B's internal BLE adapter, which does not support it. The roundtrip times measurements become significantly noisy as the distance increases, as expected, but it is particularly egregious as the payload size approaches the MTU length on the ASUS USB-BT500 tests at 9 m in Figure 3.19.

As mentioned previously, the graphs for the ASUS USB-BT500 tests, in particular in Figure 3.16, have a shape similar to that of a step function. By analyzing the data from

Figure 3.16, it is observed that each step has an average width of $36.2 \pm 0.98$ bytes, and a difference of $9.18 \pm 1.18$ ms (which is quite close to the connection interval, 7.5 ms). This likely indicates that the data is being fragmented on the L2CAP layer every ∼36 bytes (*i.e.* it is the maximum payload size that is being sent in a single connection event), and split across multiple connection intervals, instead of being transmitted in a single packet. Additionally, the fact that it increments in steps of 1 Connection Interval could also suggest that the fragmentation only occurs from one of the transmissions – either the transmission from the adapter to the nRF52-DK or from nRF52-DK to the adapter.

Moreover, this could imply that the observed delay is most likely a bottleneck on the *BlueZ* Linux *API*, delaying the transmission of data from the user level to the lower levels of the BLE protocol stack and vice versa, as the roundtrip time increases in increments of the connection interval (albeit slightly higher than it) with the payload size, which is the expected behavior, pointing to an issue on the BLE stack implementation.

Overall, it is observed that in order to maximize data throughput, the maximum supported payload size should be used to minimize the impact of the ∼12 ms delay on the transmission.

**Test 2: Bandwidth Measurement**

In this second test, the bandwidth for the BLE connection is measured by adjusting the transmission rate of the data sent, or more accurately, the time between the transmission of each packet. The test has been performed by reducing the time between transmissions from 500 ms to 10 ms, in 5 ms decrements, until the communication is stopped. In the graphs, this is converted to transmission rate in order to facilitate the interpretation of data.

For this test, the nRF52-DK board exposes 7 GATT characteristics for the *Smart box* to subscribe to, which corresponds to the maximum amount of concurrent subscriptions using *BlueZ*. This was determined empirically during the setup for the tests. The nRF52-DK then continuously changes the value of the characteristic, immediately triggering a notification to the *Smart box* – one for each characteristic. The transmission rate is increased until the limit of the connection, which eventually becomes too unstable and suddenly terminates. The payload size used on each characteristic corresponds to the maximum ATT payload observed in the roundtrip test – 20 bytes for the internal BLE adapter and 244 bytes for the ASUS USB-BT500 adapter.

Figures 3.20-3.27 show the bandwidth graphs obtained for each test configuration for each adapter. Each test configuration (*i.e.* transmission rate and distance) has been run 3 times to ensure the consistency of results, for a total of 2376 independent tests.


Figure 3.20: BLE connection bandwidth obtained using the Raspberry Pi 4B internal BLE adapter at a distance of 0 m. The maximum bandwidth achieved at this distance was 22.9 ± 7.34 kbps, achieved at 40 Hz with 43.5% packet loss.


Figure 3.21: BLE connection bandwidth obtained using the Raspberry Pi 4B internal BLE adapter at a distance of 3 m. The maximum bandwidth achieved at this distance was 24.2±7.25 kbps, achieved at 40 Hz with 41.0% packet loss.


Figure 3.22: BLE connection bandwidth obtained using the Raspberry Pi 4B internal BLE adapter at a distance of 6 m. The maximum bandwidth achieved at this distance was 24.8 ± 7.43 kbps, achieved at 33.33 Hz with 27.2% packet loss.


Figure 3.23: BLE connection bandwidth obtained using the Raspberry Pi 4B internal BLE adapter at a distance of 9 m. The maximum bandwidth achieved at this distance was 24.8 ± 6.35 kbps, achieved at 33.33 Hz with 28.9% packet loss.


Figure 3.24: BLE connection bandwidth obtained using the ASUS USB-BT500 adapter at a distance of 0 m. The maximum bandwidth achieved at this distance was 175.0 ± 6.23 kbps, achieved at 13.33 Hz with 2.13% packet loss.

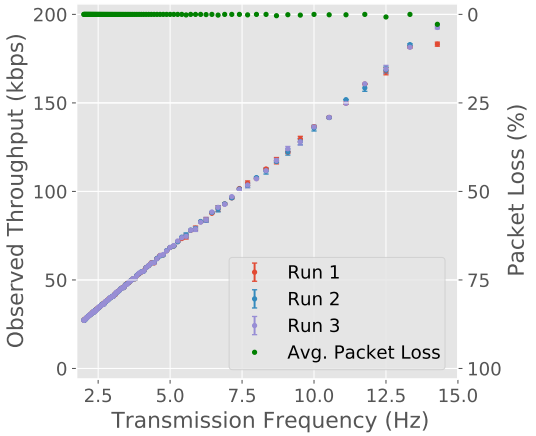


Figure 3.25: BLE connection bandwidth obtained using the ASUS USB-BT500 adapter at a distance of 3 m. The maximum bandwidth achieved at this distance was 193.0 ± 0.779 kbps, achieved at 14.29 Hz with 0.25% packet loss.

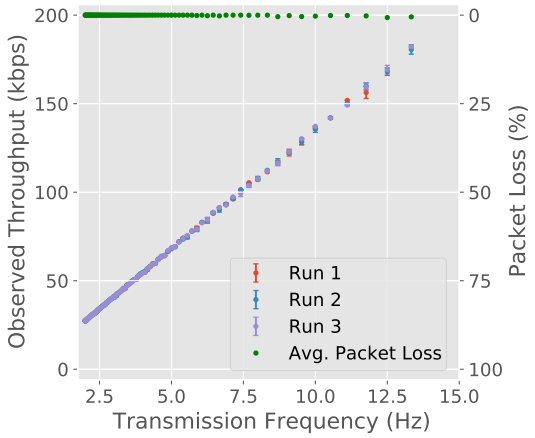


Figure 3.26: BLE connection bandwidth obtained using the ASUS USB-BT500 adapter at a distance of 6 m. The maximum bandwidth achieved at this distance was 182.0±0.0132 kbps, achieved at 13.33 Hz with 0% packet loss.

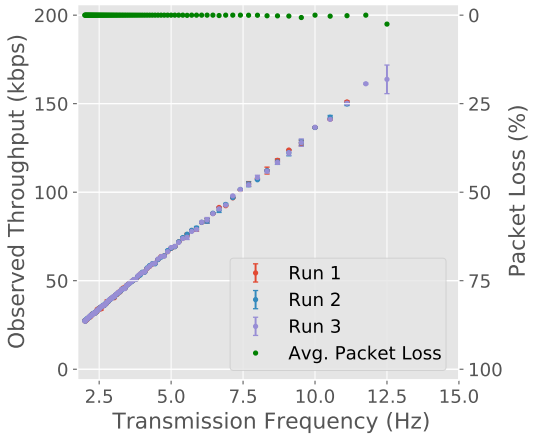


Figure 3.27: BLE connection bandwidth obtained using the ASUS USB-BT500 adapter at a distance of 9 m. The maximum bandwidth achieved at this distance was 164.0±8.08 kbps, achieved at 12.5 Hz with 2.53% packet loss.

The maximum bandwidth for a BLE connection can be estimated using the sequence diagram in Figure 3.10 and packet format in Figure 3.9. To simplify the calculation, it is assumed that the devices can transmit as much data as possible within the Connection Event, which is considered to be as long as the Connection Interval (7.5 ms). As mentioned in Section 3.2.3, all BLE connections for these tests use LE 1M modulation, which has a bit rate of 1 Mbps. Additionally, the distance between the devices is also not considered in the calculation, as the time delay associated with distance is approximately 4 ns/m [44] while the modulation time is 1 $\mu$s/bit, and therefore can be safely disregarded. Thus, the time to transmit a single BLE message is:

$$t_{TX} = (17 + p_l) \times 8 \text{ bits/byte} \times 1 \ \mu\text{s/bit} = 8 \times p_l + 136 \ \mu\text{s}, \tag{3.1}$$

where $p_l$ is the ATT payload length. Since this packet must be acknowledged, two other factors must also be considered:

1. Time to transmit the acknowledgement packet, which has an empty ATT payload and does not include the ATT header: $t_{RX} = (1 + 4 + 3 + 2 + 4 + 1) \times 8 = 120 \ \mu$s.

2. Minimum time interval between consecutive packets on the same radio channel: $t_{IFS} = 150 \ \mu$s.

Thus, the total time is:

$$t_t = t_{TX} + t_{IFS} + t_{RX} + t_{IFS} = 8 \times p_l + 556 \ \mu\text{s}, \tag{3.2}$$

With this information, a function to estimate the maximum throughput can be determined, by calculating the number of packets that can be transmitted in a single connection interval of 7.5 ms:

$$B_{max} = \left\lfloor \frac{7.5 \times 1000}{8 \times p_l + 556} \right\rfloor \times \frac{8 \times p_l}{7.5} \text{ kbps}. \tag{3.3}$$

This yields $B_{max} = 213.33$ kbps for the payload length used in the internal adapter tests and $B_{max} = 520.53$ kbps for the payload length used in the ASUS USB-BT500 tests, which are very different from those observed in the tests. Unfortunately, there are many factors which influence throughput, such as memory constraints on the BLE implementations that limit the amount of messages that can be sent in single Connection Event, and other processing delays that impact the throughput [47], which are extremely difficult to account for as the underlying causes are not evident most of the time. For example, in the roundtrip test a 12 ms delay is observed for transmission / reception of data, but it is not possible to determine exactly in which layer the delay (or delays) are occurring as the tools available for troubleshooting these issues on Linux are limited.

Additionally, packet loss can be observed as transmission frequency increases, reaching closer to the "breaking point" where the connection suddenly crashes, as expected. However, the internal BLE adapter shows a much more noticeable packet loss (particularly when the transmission frequency is greater than 15 Hz). This, coupled with the analysis of the previous tests, likely indicates that these losses are caused by limitations of the BLE stack rather than the adapter itself, *i.e.* the existing BLE implementation cannot handle these high transmission rates properly, and higher payload sizes are preferred instead. This would also explain why the packet loss is nearly 0% for the ASUS USB-BT500 tests before reaching the "breaking point", since it uses a much higher payload size.

Using these tests, it is possible to determine the adequacy of these adapters for handling the BLE communication with the *Biostickers* using the data sizes and transmission frequencies from Section 3, where it is seen that the communication with the *Biostickers* uses close to 3.593 kbps of bandwidth. Both adapters are more than capable of handling that bandwidth,

for any distance up to 9 meters, however, they cannot reach the 20 Hz transmission frequency required by the ECG data communication without significant packet loss due to limitations of the BLE stack, as discussed previously. In order to use the *Biostickers* without packet loss, the data must be grouped and sent in larger packets, thus lowering the transmission rate and increasing the packet size in order to bypass these limitations, however this approach is only possible using the ASUS USB-BT500 BLE adapter since it supports packet sizes up to 244 bytes, unlike the Raspberry Pi 4B internal BLE adapter.

Curiously, some graphs show better throughput for intermediate distances rather when the devices are next to each other, e.g. for Figure 3.20 at 0 m and 3.21 at 3 m. This is likely caused by misalignment of the devices, which can slightly improve the efficiency of the power transfer in radio transmissions (as no information about the BLE adapter antennas is known), improving the stability of the connection and thus allowing the usage of a higher transmission rate before the connection crashes. Moreover, as expected from the previous tests, ASUS USB-BT500 shows a much better throughput since it can transmit more data in the LL data packets.

### 3.2.4 Decision on the BLE adapter

From the previous tests, it is observed that the Raspberry Pi 4B's internal BLE adapter lacks the support for DLE feature, which reduces greatly the communication throughput. Additionally, this maximum throughput on Raspberry Pi 4B is achieved with an extremely high packet loss (over 40%). This suggests that to make full use of this adapter, the system and communications must be designed considering this packet loss, or reduce the BLE throughput to minimize the packet loss. The ASUS USB-BT500 on the other hand has a throughput 10 times larger than the internal adapter, with very low packet loss (which can be mitigated by slightly reducing the throughput).

Due to all the aforementioned reasons, the ASUS USB-BT500 adapter was chosen for the development of BLE data acquisition in the scope of the WoW project.

## 3.3 Summary

In this chapter, a comprehensive performance study of the different SBCs considered for the *Smart box* development was presented, as well as an extensive analysis of the BLE data

communication.

In the next chapter, the development of the next component in the proposed IoT architecture is presented – the *Smart Gateway*.

# 4 Smart Gateway Development

In the proposed architecture, the *Smart Gateway* is the central module of the system, connecting the *Smart boxes* to the HIS. It is responsible for the management of devices and their associations – *Smart box* to *Biosticker* and *Smart box* to user – managing, maintaining and storing the data that is generated by these, as well as handling any communication to and from the HIS.

Regarding the hardware platform used for the *Smart Gateway*, in the context of the WoW project, the Intel NUC NUC8i7BEH[16] is used, as seen in Figure 4.1. Table 4.1 shows the hardware specification of the Intel NUC kit used.

Figure 4.1: Intel NUC NUC8i7BEH.

Table 4.1: Intel NUC Kit NUC8i7BEH specification.

| | |
|---|---|
| **Memory** | 16 GB DDR4-2400MHz |
| **CPU** | Intel Core i7-8559U Processor (8M Cache, up to 4.50 GHz) |
| **GPU** | Iris Plus Graphics 655 |
| **Mass Storage** | 1 TB SSD |
| **Operating System** | Ubuntu Server 20.04.2 LTS |

[16] `https://ark.intel.com/content/www/us/en/ark/products/126140/intel-nuc-kit-nuc8i7beh.html`

In the next sections, a service architecture for the *Smart Gateway* is proposed in order to fulfill the aforementioned features.

## 4.1 Service Architecture

As seen in Section 2.4, there are multiple key features that form the *Smart Gateway*. The different *Smart Gateway* components are:

- **Manage devices and device associations**: The *Smart Gateway* maintains a list of all the *Smart boxes* that are managed by the system, as well as every *Biosticker* and every sensor in the *Biosticker* (which are used to indicate the respective biosignal to the HIS). The *Smart Gateway* also tracks the sensor subscriptions per *Smart box*.
- **Data anonymization**: Any private data (*i.e.* information that can be used to identify a user) that is stored in the *Smart Gateway* is anonymized in order to meet data protection regulations[17].
- **Data pre-processing**: The *Smart Gateway* processes the data as it is collected in order to clean the data before storing it indefinitely, and to detect critical conditions of the patients' state to prompt an immediate notification to the health professionals.
- **Real-time data acquisition**: The *Smart Gateway* handles the secure communications with the *Smart boxes*, acquiring the data in real-time.
- **Manage data collection**: After receiving and processing the data from the *Smart boxes*, the *Smart Gateway* stores indefinitely for long-term biomonitoring analytics.
- **HIS FHIR Integration**: The *Smart Gateway* handles the communication with the HIS. More specifically, it processes all FHIR requests from the HIS, and also transforms the acquired sensor data into FHIR messages and communicates it to the HIS.

To implement these components, the following service architecture within the *Smart Gateway* is proposed, as illustrated in Figure 4.2. The correspondence between the services and the *Smart Gateway* components is described in Table 4.2.

[17]Resolution of the Council of Ministers no. 41/2018, of 28 March, following the new General Data Protection Regulation (GDPR), approved by Regulation (EU) 2016/679: `https://dre.pt/application/file/a/114936962%20`

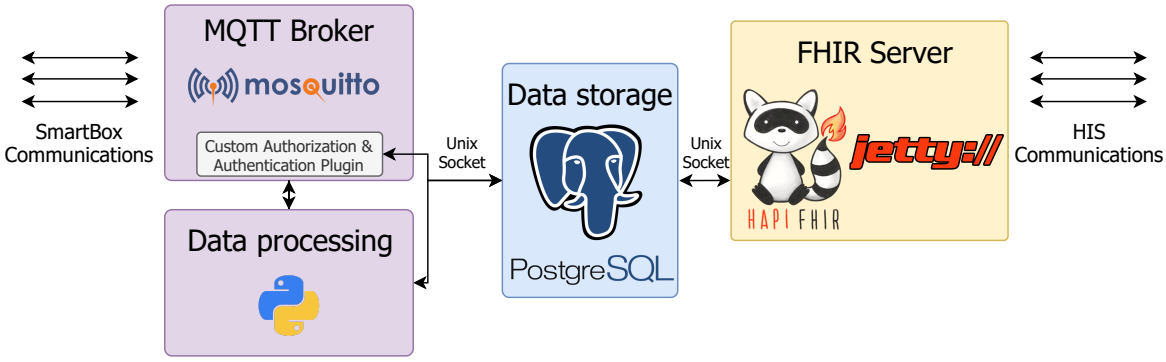


Figure 4.2: Service architecture implemented in the *Smart Gateway*. The diagram displays the different technologies used throughout the development.

Table 4.2: Correspondence between the *Smart Gateway* services and its functional components.

| ***Smart Gateway* Component** | ***Smart Gateway* Service** | **Description** |
|---|---|---|
| Real-time data acquisition | MQTT Broker | Service that handles communication with the *Smart boxes*, ensuring data encryption, authorization, etc. |
| Data pre-processing | Data processing | Data filtering and preliminary data processing. |
| Manage data collection<br>Manage devices and device associations | Data storage | Management and storage system information, such as the list of devices, permissions of each device and collected sensor data. |
| Data anonymization<br>HIS FHIR Integration | FHIR Server | Service that handles communications with the "Interoperability" layer of HIS. |

The services communicate with one another using UNIX Domain Sockets[18]. This is an interprocess communication (IPC) protocol that enables efficient communication between processes running on the same host operative system. This protocol is very efficient, compared for example to traditional network sockets [48], since all communication is handled entirely by the operative system kernel, instead of relying on the IP protocol stack, minimizing communication overhead.

The protocol can make use of the Linux file system for addressing the sockets, which means it is subject to Linux file system permissions. This allows applications to identify which process, or more accurately, the user running the process, who is attempting to establish a new connection to that application, providing a simple and secure authentication mechanism on the IPC.

[18] `https://man7.org/linux/man-pages/man7/unix.7.html`

## 4.2 Data Storage

Data storage in the *Smart Gateway* is one of the most important components of the device, as it holds the information used by all services in the *Smart Gateway*. Given the importance of this component, it is crucial to use a solution which offers reliability above all, with proved performance for our use case.

As discussed in Section 2.2.4, NoSQL databases are appealing for IoT applications, since these can handle unstructured or semi-structured data and generally perform better than traditional SQL databases as the amount of data stored increases. However, these systems are not adequate for a relational data model, which is required to enforce consistent and logical representation of information. For this reason, a traditional Relational Database Management System (RDBMS) has been deployed for data storage in the system.

Out of the different RDBMSs available in the market, PostgreSQL stands out due to its overall performance and scalability [49]. Additionally, it is one of the most popular RDBMS [50], meaning it also has significant community support.

With this in mind, PostgreSQL has been chosen as the data storage technology in the *Smart Gateway*. PostgreSQL[19] is an advanced, enterprise-class, and open-source RDBMS. It has over 30 years of active development by the open source community, earning a strong reputation for its reliability, feature set and robustness.

### 4.2.1 Database Schema

Figure 4.3 contains the database model implemented in our PostgreSQL database. It describes all information that is contained in the *Smart Gateway*, the relations within that data, organized according to how that information is used (*i.e.* the service / functionality it is associated with). The data stored in the system can be categorized into 5 distinct groups:

1. **System data** – Information about the devices which are managed by the system: the *Smart boxes*, the *Biostickers* and the sensors in each *Biostickers*.

2. **MQTT related data** – Information about the MQTT clients and their permissions.

[19] `https://www.postgresql.org`

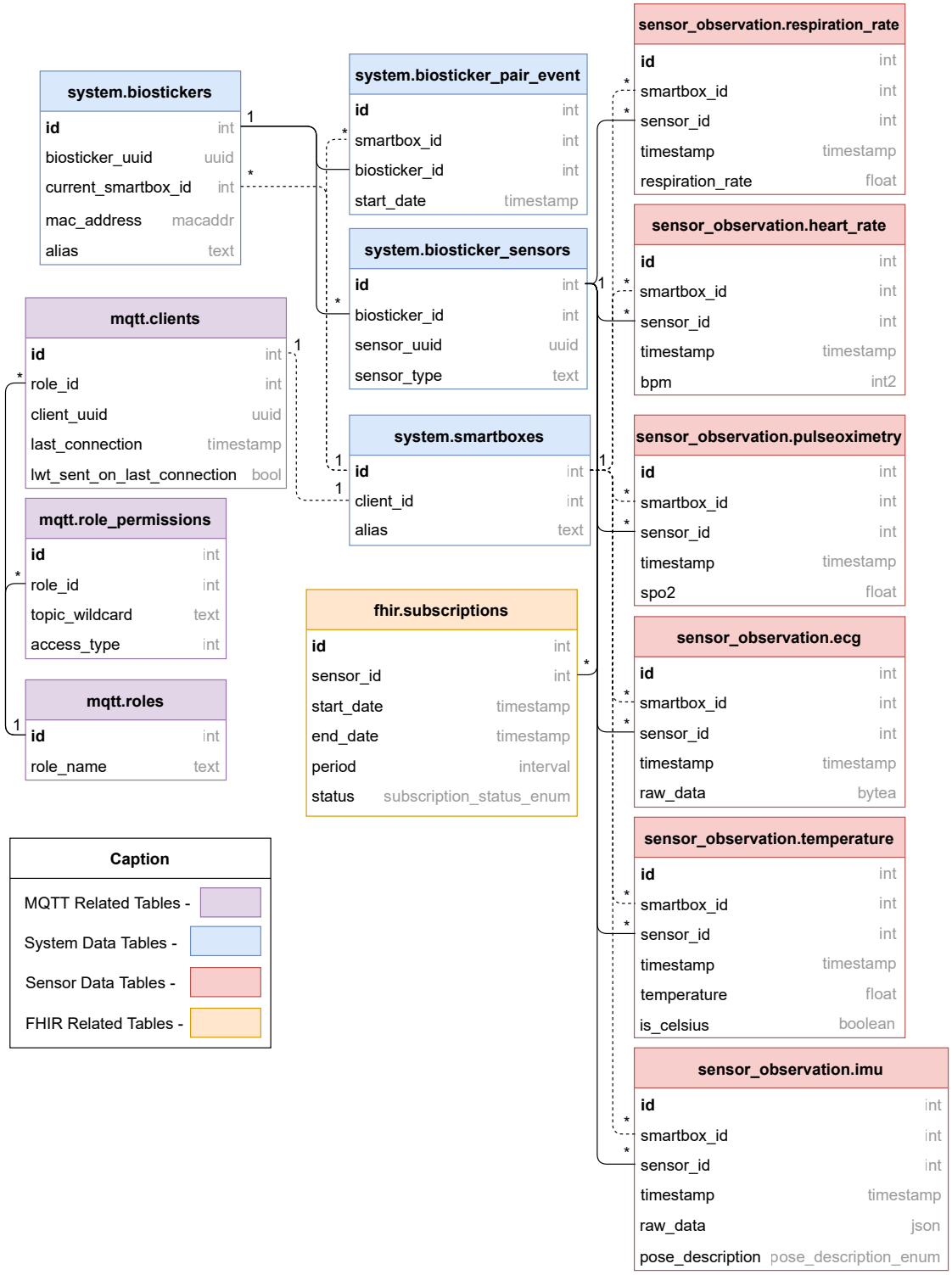


Figure 4.3: Database model implemented in the *Smart Gateway*. The bold text in the diagram is used to denote the Primary Key (PK) of each table. The relationships between entities are indicated with a line and using the symbols "*" for many and "1" for one.

3. **Sensor observation data** – Biosignals measured and communicated by the *Smart boxes*.

4. **FHIR data** – Data related with FHIR communications, such as the subscription

requests from the HIS to communicate the acquired sensor measurements.

5. **Stored procedures** – Custom subroutines that define the operations used by other services (*e.g.* the MQTT broker) to interact with the stored data (insertions, deletions, searches, etc.).

In the next sections, the structure of the data within each of these groups is explored in greater detail.

### System data

Figure 4.4 describes the components or entities of the database model that depict system information. The data model is designed with flexibility in mind, allowing each *Smart box* to be associated with any number of *Biostickers*, and each *Biosticker* to have any number of sensors associated to it.

Each sensor is uniquely identified by an UUID when communicating the sensor measurement to the HIS. As the names suggest, the "system.biostickers" table contains the list and details of all *Biostickers*, "system.smartboxes" table contains the list and details of all *Smart boxes*, "system.biosticker_sensors" contains the list and details of all sensors of all *Biostickers*. The "system.biosticker_pair_event" table is used to track the history of which *Biostickers* were or are currently associated with a specific *Smart box*.

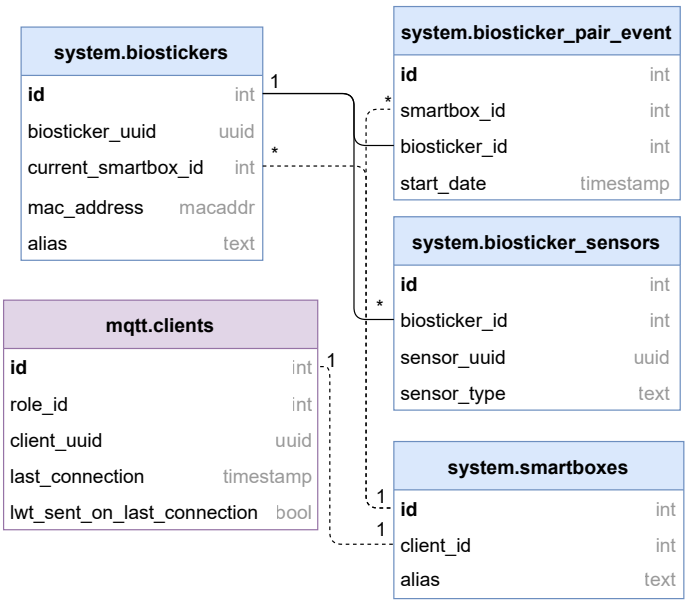


Figure 4.4: Components of the database model used to describe system information.

### MQTT related data

Figure 4.5 describes the information relevant for MQTT communications, mostly related with security. To ensure that each device only has access to allowed resources, the system implements a role-based access control (RBAC) policy. In this type of access control, the system allows or revokes access to resources according to the role of the device, meaning that all devices with a given role share the same list of permissions. The permissions for the RBAC policy contain 3 properties: the ID of the role it applies to, the topic name, and the level of access (PUBLISH, SUBSCRIBE and/or READ) to be granted (or revoked), as seen in Figure 4.5. READ access in this context is the ability to receive messages from the broker when subscribed to that topic. SUBSCRIBE access is the ability to issue a subscription request, and PUBLISH the ability to publish messages.

In context of the WoW project, the following roles are used:

- *Smart box* role: Indicates that the MQTT client is a *Smart box*.
- "Pyservice" role: Indicates that the MQTT client is actually the data pre-processing service, also contained in the *Smart Gateway*.
- Developer device role: Indicates that the MQTT client is a developer device, used solely for debugging purposes.

The "mqtt.roles" table contains the different RBAC roles for the MQTT communication and "mqtt.role_permissions" table lists the permissions available to each role using MQTT topic wildcards. The "mqtt.client" table lists the clients and their properties, such as their UUID, the timestamp of their last connection, or a *flag* to indicate if the communication failed during the last communication.

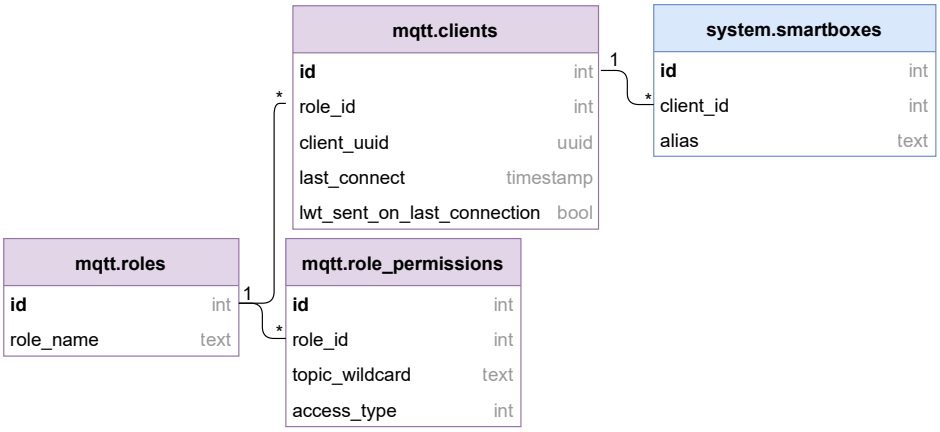


Figure 4.5: Components of the database model used to describe MQTT information.

### Sensor observation data

Figure 4.6 describes the information of the sensor measurements collected over time. Each signal measurement is associated with the sensor that measured it and the *Smart box* that is associated to that sensor, or more accurately, associated to the *Biosticker*, at the moment of the observation.

The database model has one table for each type of biosignal measured in the WoW project (temperature, ECG, etc.). The properties of the table are defined according to the structure of the data that is acquired by the *Smart box*, which are detailed in Section 3. The "pose_description" field in "sensor_observations.imu" table is a text representation of the different body poses according to an international health standard[20].

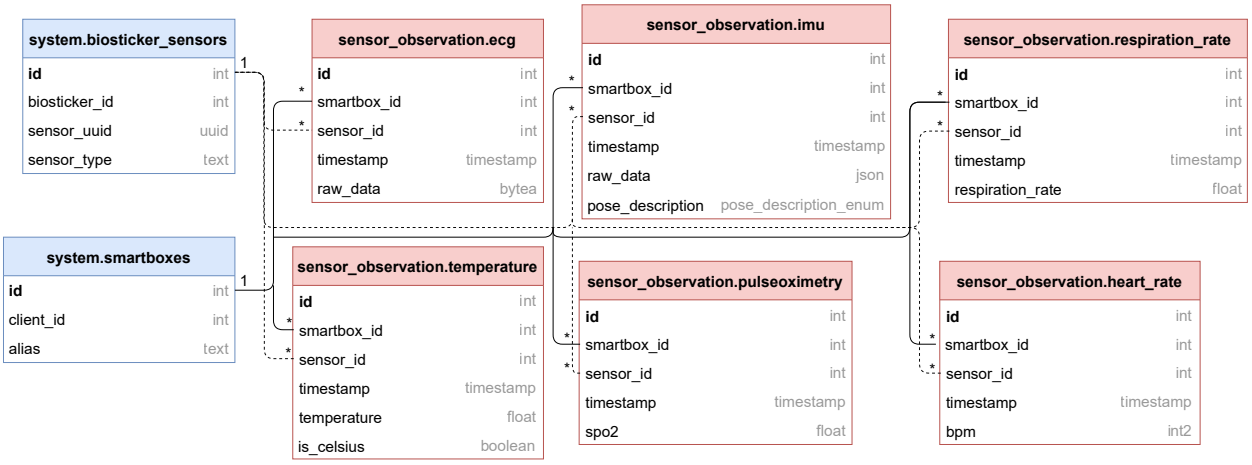


Figure 4.6: Components of the database model used to describe sensor measurements.

[20] `https://loinc.org/8361-8/`

### FHIR data

Figure 4.7 describes the information associated with the FHIR communications. Currently, the only information that is stored in the database is the list of subscription requests sent from the HIS. The "status" field in the "fhir.subscription" table indicates the status of the subscription request (active, completed, revoked, etc.) and should be a text value that matches its equivalent in the FHIR enumeration [51].

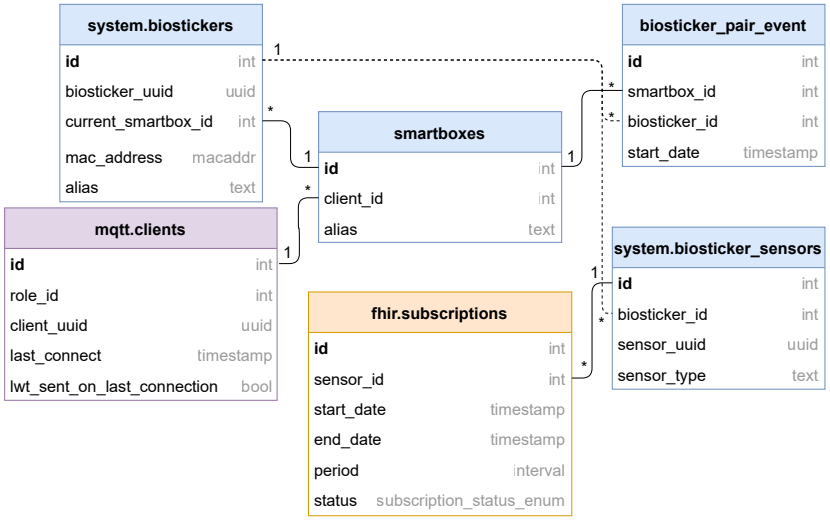


Figure 4.7: Components of the database model used to describe information used for FHIR.

### Stored Procedures

Since the data storage implemented in the *Smart Gateway* is a RDBMS, services that access the database use Structured Query Language (SQL) to perform requests, such as retrieving or inserting data. In order to maximize the performance of our data storage solution "Stored Procedures" are implemented, which are custom subroutines that are stored in the RDBMS. These procedures are pre-compiled SQL statements, which are simply a set of instructions that perform a given task, that are defined in the RDBMS, and can greatly improve the performance of these systems since these:

- Reduce significantly the amount of data that is exchanged – instead of sending a request with a complex SQL query to the database, the application sends a request for the execution of a subroutine along with its parameters, thus reducing the size of the request and the time it takes to interpret it.
- Reduce significantly the amount of data that is exchanged – as these SQL statements are optimized when pre-compiled.

- Increase the security and robustness of the database system – since the SQL statements are pre-compiled, this mitigates possible SQL injections attacks [52], also providing us with the ability to restrict the permissions of the applications that access the RDBMS to execute only certain subroutines, instead of allowing them to perform general SQL requests.

In total, over 33 procedures have been implemented in the data storage. These procedures are very simple, *e.g.* the "system.insert_temperature_observation" procedure is implemented as follows:

```
1  CREATE PROCEDURE system.insert_temperature_observation(
       ref_sensor_uuid uuid, ref_timestamp timestamp with time zone,
       ref_temperature double precision, ref_is_celsius boolean)
       LANGUAGE plpgsql SECURITY DEFINER
2  AS $$
3  DECLARE ref_sensor_id integer;
4  ref_smartbox_id integer;
5  BEGIN
6  SELECT system.biostickers.smartbox_id,
7    system.biosticker_sensors.id INTO ref_smartbox_id,
8    ref_sensor_id
9  FROM system.biosticker_sensors
10   INNER JOIN system.biostickers ON (
11     system.biosticker_sensors.biosticker_id = system.biostickers.id
12   )
13 WHERE system.biosticker_sensors.sensor_uuid = ref_sensor_uuid;
14 INSERT INTO sensor_observations.temperature (
15     smartbox_id, sensor_id, timestamp, temperature, is_celsius)
16 VALUES(
17     ref_smartbox_id, ref_sensor_id, ref_timestamp,
18     ref_temperature, ref_is_celsius);
19 END;
20 $$;
```

## 4.3 Connection to the Smart boxes

As previously mentioned, the connection to the *Smart boxes* is performed via MQTT. In this system, the MQTT broker is contained within the *Smart Gateway*, and is the service responsible for ensuring the communication between the *Smart boxes* and the *Smart Gateway*.

To implement this broker, the open-source Eclipse Mosquitto [53] has been used. Mosquitto is a lightweight MQTT broker that supports the MQTT protocol versions 5.0, 3.1.1 and 3.1 and is widely used by the community, making it a fitting solution for the WoW project. However, in order to implement all security features required, its existing functionality must be expanded upon, this is further discussed in Section 4.3.2.

Firstly, the intricacies of the MQTT communication between the *Smart box* and *Smart Gateway* must be defined. To this end, a complete specification is proposed, detailing the all security measures implemented, the format for the messages exchanged in the communication and the different endpoints (or topics) used.

### 4.3.1 Proposed MQTT Specification

The MQTT standard to be used in all communications is the latest revision[21], MQTT 5.0. Additionally, to authenticate and encrypt transmissions between devices, the communication is secured with TLS v1.2[22] and each MQTT client must have its own X.509 V3[23] certificate and UUID to uniquely identify it. The aforementioned certificate must have the client UUID in the "Common Name" field, which is used to ensure that the certificate is issued to that specific MQTT client.

Regarding security, as mentioned previously, the system uses a role-based access control (RBAC) policy to authorize access to the MQTT topics. This means that devices of the same type (*e.g. Smart boxes*) share the same permissions. Nonetheless, the access of the devices can be restricted to its own individual topics by including a client UUID wildcard in the topic name when assigning the permission.

[21] `https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html`
[22] `https://tools.ietf.org/html/rfc5246`
[23] `https://tools.ietf.org/html/rfc5280`

For example, a permission that grants PUBLISH access to the "smartbox/%c/temperature" topic to all *Smart boxes* can be defined, where "%c" is a wildcard for the client UUID. This means that a *Smart box* with client UUID "1" can publish a message to the topic "smartbox/1/temperature", but cannot publish to "smartbox/2/temperature".

### Message Format

In order to promote interoperability, all messages exchanged in the MQTT communication must follow the JSON data format. Additionally, these must have the following structure:

```
{
    "client_id": client_uuid,
    "timestamp": timestamp,
    "message_type": message_type,
    "payload": {
        //...
    },
}
```

where *client_id* is the UUID of the MQTT client, *timestamp* is the UNIX timestamp[24], and *payload* contains the actual content of the message that is associated to the *message_type*. The field *message_type* defines what type of message it is, and must be one of the following:

- "MEASUREMENT_TEMPERATURE": Indicates that the message is a temperature measurement.
- "MEASUREMENT_IMU": Indicates that the message is an IMU measurement.
- "MEASUREMENT_HR": Indicates that the message is a heart rate measurement.
- "MEASUREMENT_ECG": Indicates that the message is an ECG measurement.
- "MEASUREMENT_PULSEOXIMETRY": Indicates that the message is a pulse oximetry measurement.
- "MEASUREMENT_RESPIRATION": Indicates that the message is a respiratory rate measurement.

For the payload formats of each of these messages, the reader is referred to Appendix A.

[24] https://www.unixtimestamp.com/

#### Data endpoints

To communicate the sensor data, the *Smart box* must publish to different endpoints, depending on the type of sensor data that is transmitted:

- **Temperature** data: "smartbox/%c/temperature".
- **Inertial Measurement Unit (IMU)** data: "smartbox/%c/imu".
- **Electrocardiogram (ECG)** data: "smartbox/%c/ecg".
- **Pulse Oximetry** data: "smartbox/%c/pulseoximetry".
- **Heart Rate** data: "smartbox/%c/heartrate".
- **Respiration Rate** data: "smartbox/%c/respiration".

### 4.3.2 Authorization and Authentication Plugin

One of the major flaws of Mosquitto is that it does not supply proper dynamic authentication and authorization mechanisms for the MQTT communication out-of-the box. By default, the list of MQTT authorized clients and their permissions are static, defined by a configuration file which is processed at the start of the program [53]. In order to implement proper security measures, Mosquitto exposes an extensive plugin API [53] that covers authentication, access control, and message inspection and modification; which is used to develop our own custom plugin to fulfill the security requirements for the WoW project. The code for the plugin can be found here[25].

The plugin works by intercepting authorization and authentication requests from the MQTT broker, and validating the information in them.

Figure 4.8 describes how a client is authorized by the MQTT broker. The process starts with the X.509 certificate validation at the TLS layer. If the certificate is valid, Mosquitto proceeds by sending an authentication request for that client to the plugin. In the plugin, X.509 certificate information is validated, in particular the "Common Name" field, and ensure the client is registered in the database.

[25] `https://github.com/WoW-Institute-of-Systems-and-Robotics/mosquitto-auth-plugin`

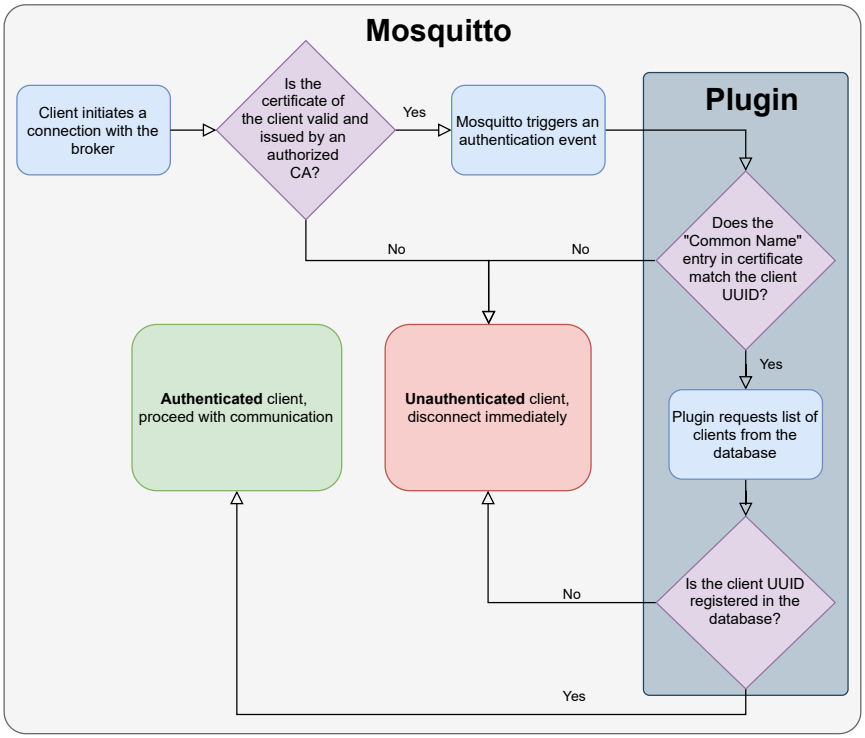


Figure 4.8: Flowchart describing how a MQTT client is authenticated by the MQTT broker.

Figure 4.9 describes how a client's request is authorized using this plugin. Since the client is already authenticated, Mosquitto proceeds by sending an authorization request for that client to the plugin. The plugin requests the list of permissions associated with the client role, and then checks if any permission on that list explicitly grants PUBLISH access to the topic.

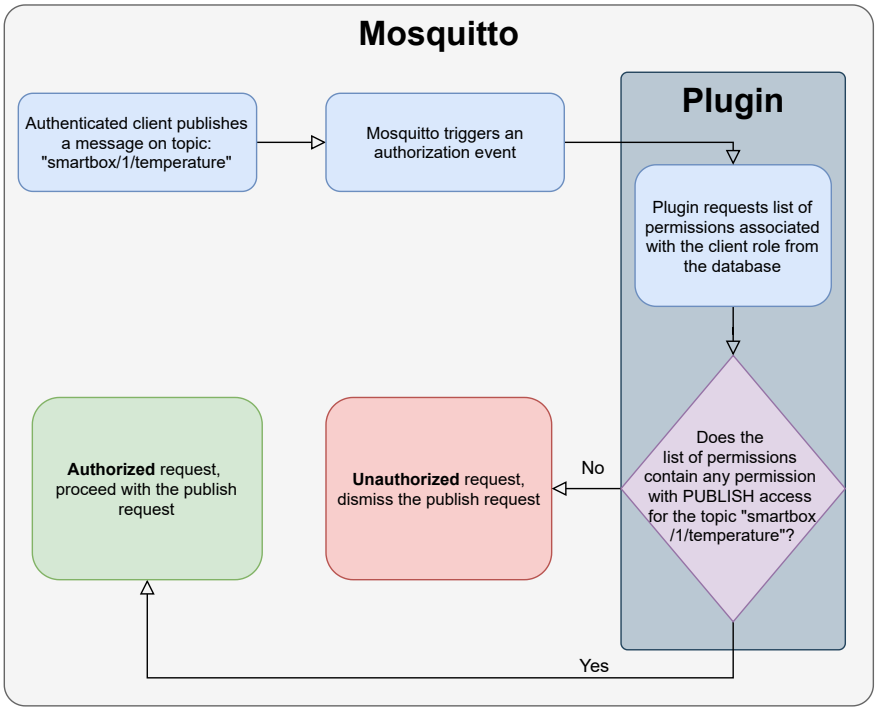


Figure 4.9: Flowchart describing how an authenticated MQTT client's request is authorized by the MQTT broker.

## 4.4 Data pre-processing

The Data pre-processing service is used to process the incoming data from the *Smart boxes* in real-time. It subscribes to incoming MQTT messages using a MQTT client with superuser privileges, that grants access to all topics. It validates the MQTT messages according to the message formats specified in Section 4.3.1, filters any irrelevant information, and stores it in the database. Currently, it does not apply any data analytics to detect critical conditions. The code for the service can be found here[26].

Figure 4.10 shows how incoming data is processed by the service.

[26] `https://github.com/WoW-Institute-of-Systems-and-Robotics/gateway_pyservice`

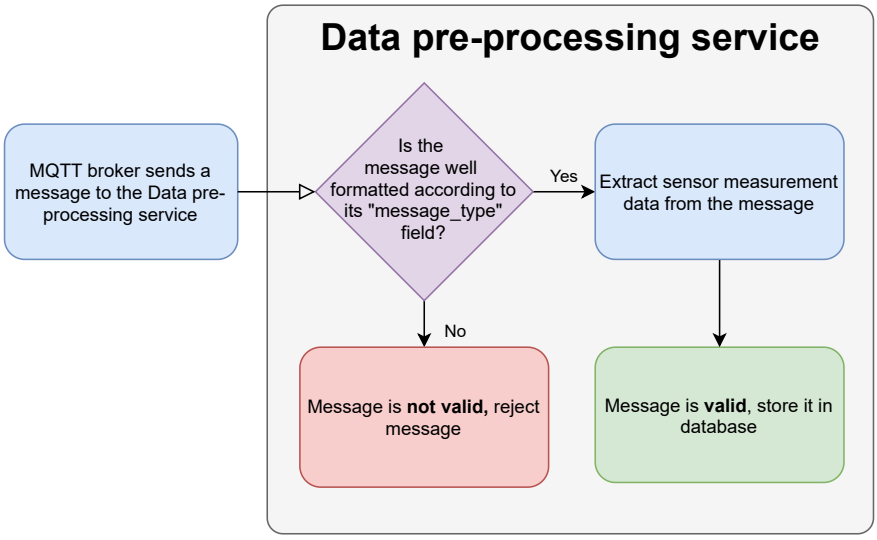


Figure 4.10: Flowchart describing how incoming MQTT messages are processed by the data pre-processing service.

## 4.5 HIS FHIR Integration

The HIS FHIR Integration service is used to manage the communication to and from GlobalCare HIS. The service must have a FHIR HTTP server [51] capable of handling requests from the HIS, as well as transform the sensor measurement data into FHIR messages and communicate it to the HIS.

Out of the open-source implementations of the FHIR specification available[27], the HAPI FHIR Java library [54] is one of the longest supported FHIR implementations, with over 18 years of active development. The project is backed and maintained by Smile CDR[28], a health technology company with a long-standing reputation in the health IT field. The HAPI FHIR library also provides a simple and intuitive API to interact with FHIR resources [51], the objects used to represent any data in the protocol, handling all data parsing or serialization of data into FHIR resources and vice versa.

For these reasons HAPI FHIR Java library has been used to implement the FHIR server. The software library provides several mechanisms to build FHIR HTTP servers. Although other models are available, the Plain Server model [54] has been used to develop the FHIR server since it just provides the bare-bones structure to build the API, and not a full-fledged

[27] `https://confluence.hl7.org/pages/viewpage.action?pageId=35718838`
[28] `https://www.smilecdr.com`

implementation with its own storage and functionality implemented, making it very flexible to work with. Using this model requires only the implementation of how the FHIR resource interactions translate to interactions with the data storage solution to create our own FHIR server, while the HAPI FHIR handles all the HTTP processing, as well as parsing and serialization of data into FHIR resources. Even though FHIR supports both Extensible Markup Language (XML) and JavaScript Object Notation (JSON) data formats, only the JSON data format is used for representing the FHIR resources in the FHIR server.

The HAPI FHIR Plain Server implementation is based on Java Servlet 3.1 API[29]. Servlets are applications that are hosted on web servers, used to extend their capabilities. This means that in order to have a functional FHIR server, a web server capable of hosting the HAPI FHIR Plain Server servlet is required. Since Eclipse Jetty9[30] is the web server used on the HAPI FHIR documentation, and is a relatively popular server (being used by Facebook, Google, Yahoo, etc.), it has been chosen for the development of the HIS FHIR Integration service. The code for the service can be found here[31].

In order to prepare the system for deployment in the first hospital trials, the development team has decided to use a pre-defined list of the subscriptions which remains static for the duration of the execution of the service. Additionally, the authentication protocol used on the FHIR communication is Basic Authentication (using a static username and password), instead of the more secure option – *OAuth2* [32] – that was originally planned for implementation.

### 4.5.1 FHIR Server

The servlet is composed by three major components:

- "Resource Providers" – defines the interactions with the FHIR resources over HTTP which are supported by our FHIR server. It also invokes the CRUD operations (create, read, update and delete) over our arbitrary data store through the "Database Handler".
- "Database Handler" – defines how the FHIR server connects to the data storage solution, which implements and exposes the methods used by the "Resource Providers" to

[29] `https://docs.oracle.com/javaee/7/tutorial/servlets.htm`
[30] `https://www.eclipse.org/jetty/`
[31] `https://github.com/WoW-Institute-of-Systems-and-Robotics/gateway_fhir_server/`
[32] `https://oauth.net/2/`

interact with the data. This component is also responsible for translating the data as it is stored in the data storage solution into valid FHIR resources and vice versa.

- "Subscription Handler" – handles the scheduling and transmission of sensor data to the HIS using FHIR Observations [51].

The Resource Providers define how and what resources are supported by the FHIR server. Currently, the only interaction that is implemented is a *read* operation [51] on FHIR Device resources [51]. FHIR Devices are the resources used to represent the *Smart box* and *Biosticker* sensors. Figure 4.11 shows how this interaction is processed by the FHIR server.

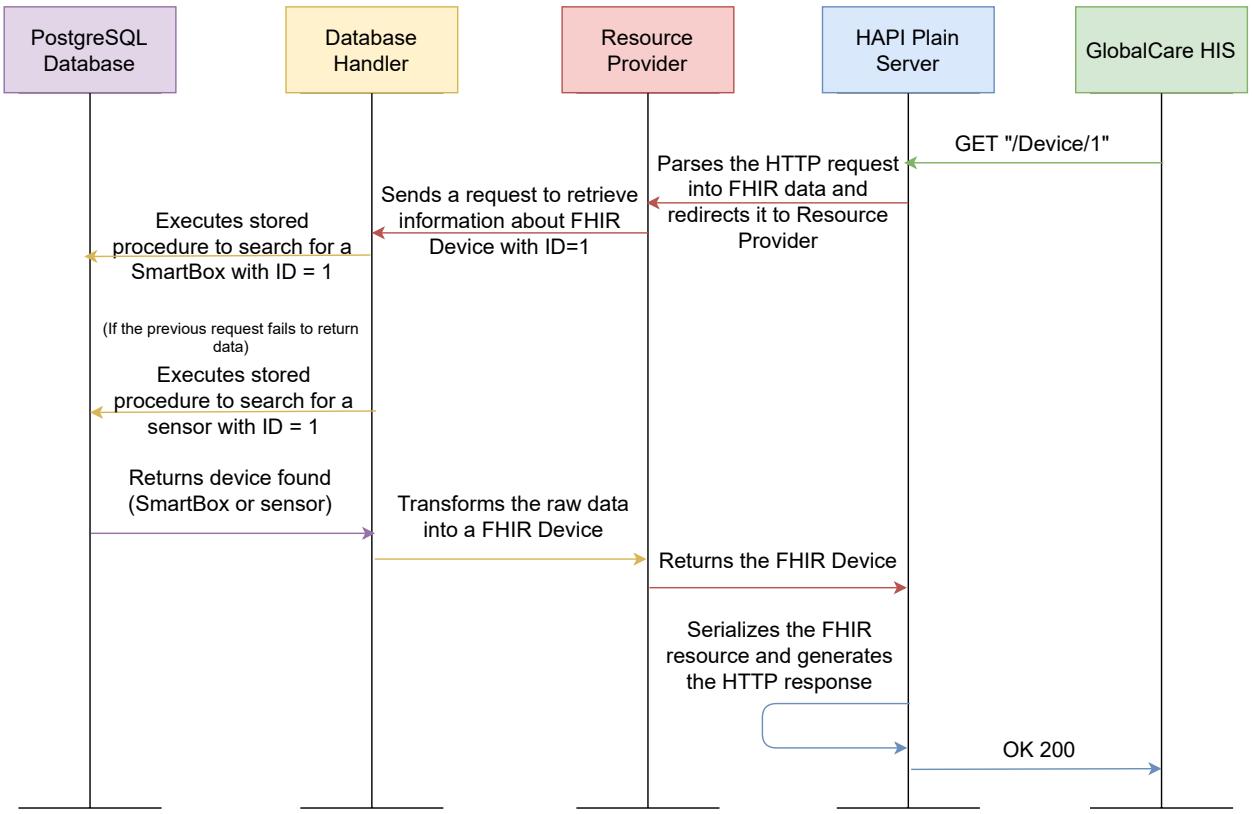


Figure 4.11: Sequence diagram describing the *read* interaction on FHIR Device resource. Although this is describing an interaction on the Device resources, the sequence diagram of a *read* interaction on any other FHIR resource should be very similar to this one.

As mentioned, the FHIR Device resource is used to represent both the *Smart box* and the *Biosticker* sensors in the FHIR protocol. To distinguish if a Device resource is a sensor or is a *Smart box*, the resource uses the field "Device.identifier" to report the device's UUID, and the field "Device.type" to describe the type of device using codes from an international

health code-set[33]. Additionally, sensors are considered "child" Devices that must always have a parent Device associated to them. This means Device resources for sensors define the field "Device.parent", which must contain a reference to a *Smart box*. The reader is referred to Appendix B for the JSON representation of the FHIR resource for the *Smart box* and for a sensor.

The reader is referred to Appendix B for the JSON representations of the FHIR resources exchanged with the HIS.

The Subscription Handler is the component responsible for handling data subscription requests from the HIS. To set up the subscriptions, after the FHIR server initializes, the Subscription Handler sends a request to the Database Handler to retrieve the list of all active subscriptions. It then schedules tasks using the Quartz Scheduler library[34] according to the information specified in the subscription data, in order to trigger notification events periodically, as seen in Figure 4.12.

The process is triggered by the Scheduler when there is a scheduled notification task at that given time. The FHIR server parses the sensor data into its FHIR representation, the FHIR Observation resource, and then bundles it, using Bundle resource [51], with the FHIR Device resources of the sensor and *Smart box* associated to that sensor. This is necessary because the "Interoperability" layer on the GlobalCare HIS does not hold information regarding the associations between the sensors and the *Smart box*, and the associations with patients are performed in regard to the *Smart box*, not the sensor. Instead, it relies on the *Smart Gateway* to send that data, along with the measurement to properly process it.

[33] `https://www.snomed.org`

[34] `http://www.quartz-scheduler.org/`

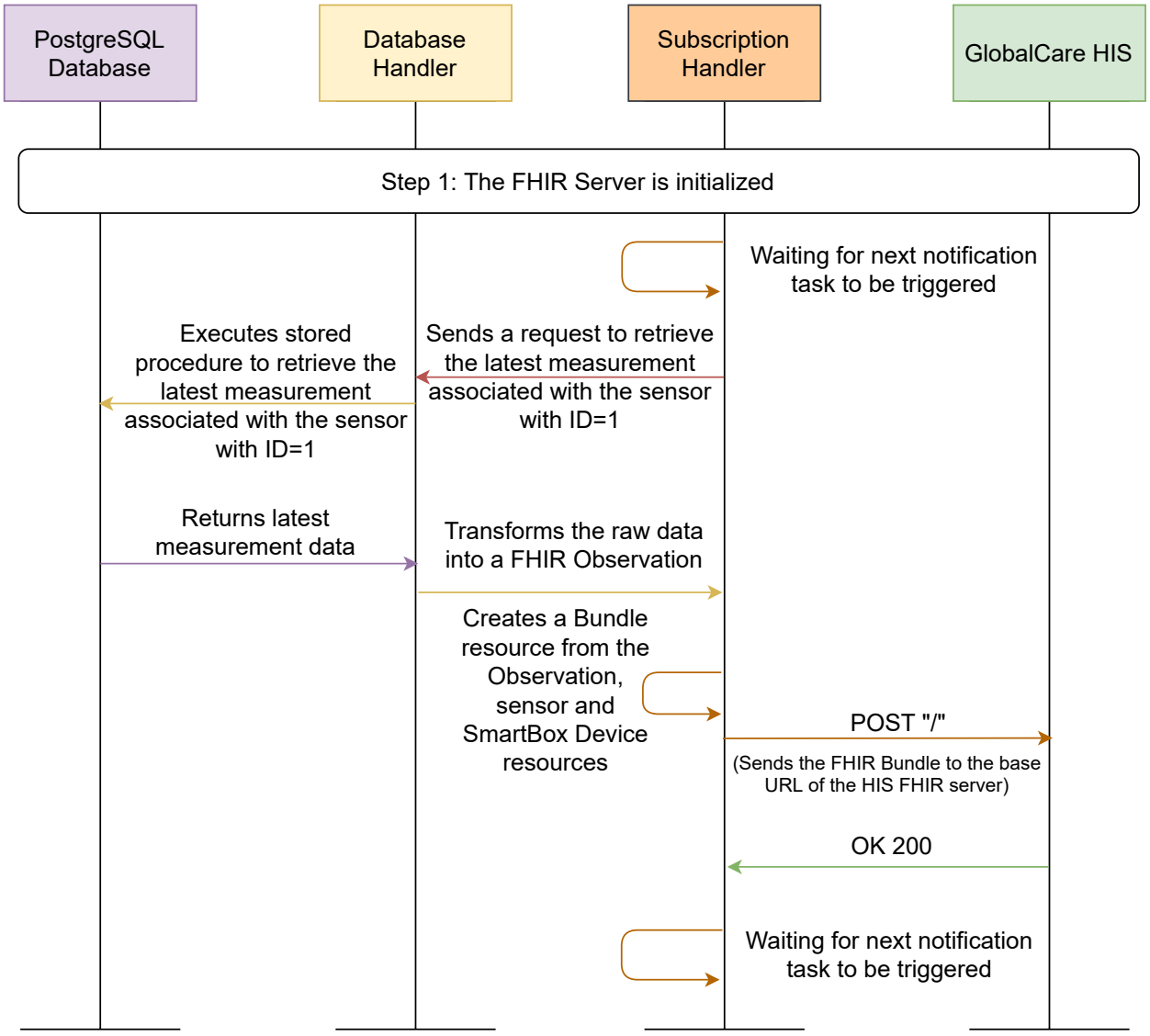


Figure 4.12: Sequence diagram describing the communication of sensor data to the HIS.

## 4.6 Summary

In this chapter, the different components which form the *Smart Gateway* are presented. Next, the performance of the proposed solution is evaluated through a hospital trial and controlled lab tests.

# 5 Experimental Validation

After developing and analyzing the different IoT system components, it is time to evaluate its overall performance in a real-world scenario. In this chapter, the results of the trials performed on the overall IoT system are presented and discussed.

## 5.1 Hospital Pilot

For the hospital trial, the proposed IoT system has been deployed in a clinical facility within Centro Hospitalar e Universitário de Coimbra (CHUC), during which two volunteers have been continuously monitored using the system. A *Smart box*, one *Biosticker* and one oximeter are assigned to each volunteer for the duration of the trial. The oximeter is attached to the patient's right-hand index finger recording pulse oximeter data and the *Biosticker* is attached to the patient's chest recording the other biosignals (body temperature, ECG, etc., as discussed in Section 3). The patients remained in bed for the entirety of the trial, always keeping the *Smart box* at most 5 meters away from the patient, as shown in Figure 5.1. The Wi-Fi network used to connect the *Smart box* to the *Smart Gateway* has been provided by the hospital IT, and is also being used concurrently by the researchers during the study. Additionally, each device is assigned a fixed IP to facilitate the deployment of the infrastructure, and ensure the devices are able to communicate with one another at all times.

The acquisition rates of the sensors are as defined in Section 3. As mentioned in the previous chapter, for this first trial, the subscription rates for the FHIR communications have been previously established, corresponding to 1 minute intervals for all sensors, *i.e.* the latest measurement of each biosignal is communicated every minute to the HIS.

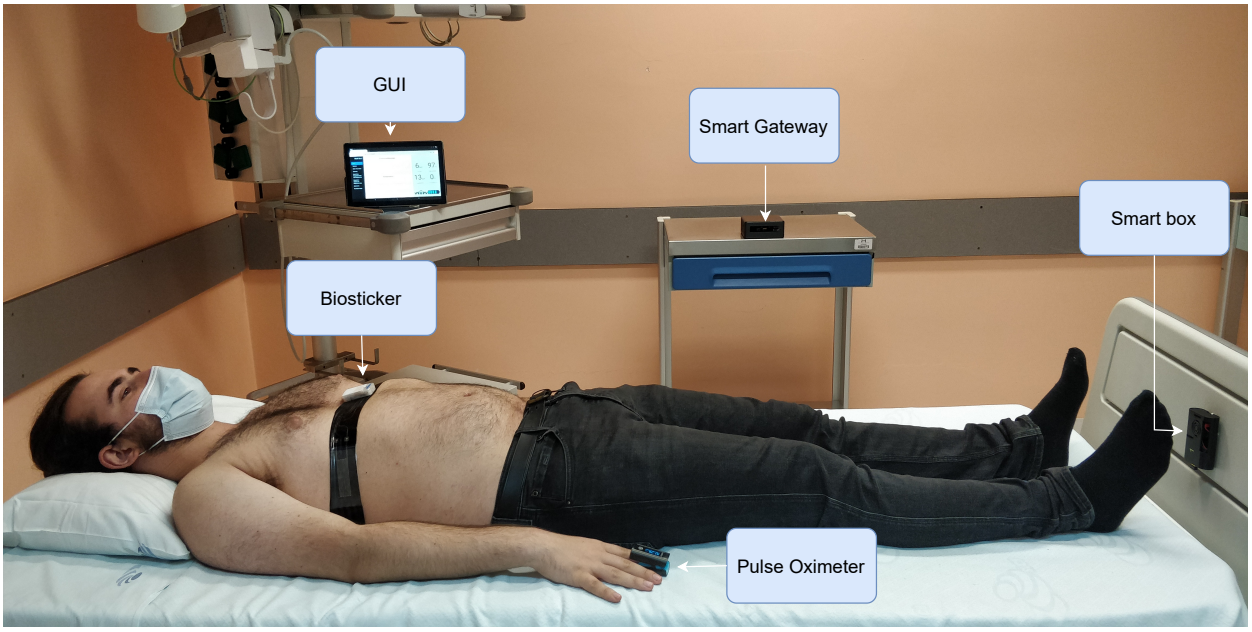


Figure 5.1: Conceptual illustration of the system components within a medical facility.

In this trial, the objective is to evaluate the stability and reliability of the entire infrastructure in a real-world scenario. To this end, the bandwidth used by the MQTT and FHIR communication protocols are analyzed to evaluate the reliability of the system, in order to check for interruptions of the communication and observe if there are any issues with the communication; and the system resource usage (CPU and RAM) is also monitored to evaluate the system's stability. To evaluate the performance of the proposed system, the following performance metrics were defined and measured throughout the tests:

- MQTT bandwidth – Rate of data exchanged between all *Smart boxes* and *Smart Gateway*;
- FHIR bandwidth – Rate of data exchanged between the *Smart Gateway* and HIS;
- Resource usage of *Gateway* services – CPU and RAM usage of each *Gateway* service;

In the next section, 2 hours of continuous monitorization tests are analyzed and discussed.

### 5.1.1 Results and Discussion

Figure 5.2 shows a boxplot of the MQTT payload sizes for each type of biosignal message. The deviation in the payload lengths is caused by variations of the number of digits in the numeric entries sent in the messages (*e.g.* acceleration or gyroscope values in the IMU), which should result in using more or less characters when the data is serialized into the JSON

payloads sent to the MQTT broker. For the heart rate, respiration rate and pulse oximetry messages, due to the nature of these biosignals the number of digits does not change, so the payload size remains constant throughout the tests. Taking this information into account, together with the acquisition rate for each biosignal as defined in Section 3, the estimated bandwidth is expected to be close to $175.7 \pm 0.2$ kbps.

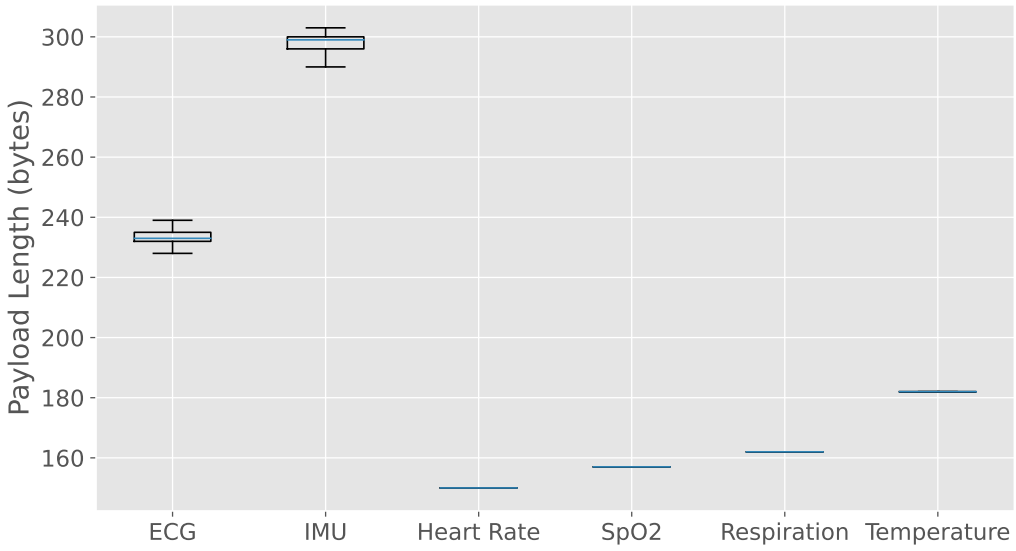


Figure 5.2: MQTT payload sizes measured for each type of biosignal sensor message during the hospital trial.

The following graphs have been obtained using the data collected in the hospital trial using 2 *Smart boxes* and 1 *Smart Gateway*. Figure 5.3 and 5.4 show the measured MQTT and FHIR bandwidth respectively, averaged over 1 min. Figure 5.5 and 5.6 show the measured CPU and RAM usage.

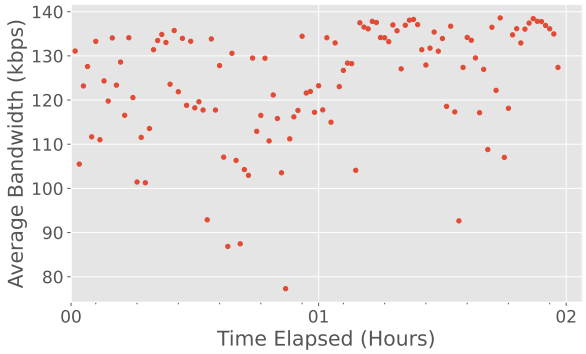


Figure 5.3: Average MQTT bandwidth usage measured over time during the hospital trial.

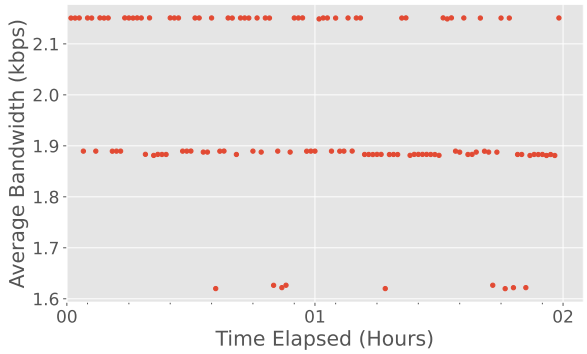


Figure 5.4: Average FHIR bandwidth usage measured over time during the hospital trial.

Yet, the measured bandwidth data is very different from the abovementioned value – with an average value of $124.1 \pm 12.8$ kbps. Not only it is $\sim$30% lower than the estimated value, the bandwidth results are very sparse as seen by the standard deviation of 12.8 kbps, which corresponds roughly to 10 messages per second. Despite our efforts, it is not possible to determine the exact nature of this deviation since there are no records of the MQTT transmissions on the *Smart box* side.

There are several factors which influence the MQTT bandwidth, any of which (or combination of which) could be causing this issue:

- The custom *Mosquitto* plugin intercepts messages to authenticate and validate the *Smart boxes*. This introduces an additional processing delay that could impact the measured bandwidth.
- The *Smart boxes* continuously receive data from the *Biostickers*, so any interruption on the communication between these devices inevitably would reduce the amount of messages sent. During the hospital trial, numerous BLE connection issues were reported, so this is likely one of the main causes for the observed variance.
- Since the *Smart boxes* are connected using Wi-Fi, and since the network was being actively used by researchers throughout the trials, there could be fluctuations in the transmission of the data caused by network constraints. This could contribute to the issue at hand, but should not be significant enough to take into account.

Due to these irregularities, the FHIR bandwidth is impacted, as seen in Figure 5.4. The graph shows the data arranged in different "levels", each corresponding to the successful transmission of a certain amount of messages. These levels are discretized as the subscriptions are only triggered once a minute. In this case, the maximum level (at $\sim$2.16 kbps) corresponds to sending all messages in that minute, $\sim$1.92 kbps when one message is not sent in that minute, and $\sim$1.6 kbps when two messages are not sent instead. The messages are not being sent because there is no new information in the data storage, which may be caused by the issues previously referred. Additionally, all messages sent by the *Smart Gateway* to GlobalCare HIS have been confirmed on the HIS side as well, thus validating the FHIR service developed.

Regarding CPU and RAM usage, it is observed to be very low across all services running on the *Smart Gateway* (the hardware specification can be found on Table 4.1) with less

than 10% CPU usage and 2% RAM usage overall, meaning that the service architecture is fairly efficient. This also provides ample resources for deploying additional analytics services locally, instead of relying on cloud services.

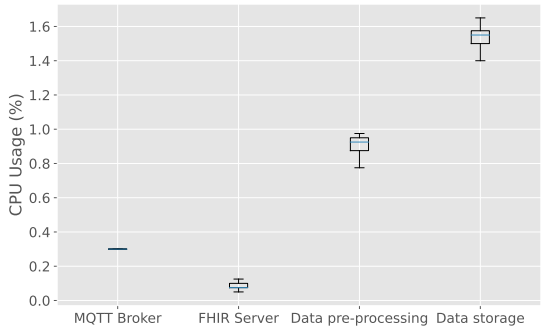


Figure 5.5: CPU usage of each *Smart Gateway* service measured over time during the hospital trial.

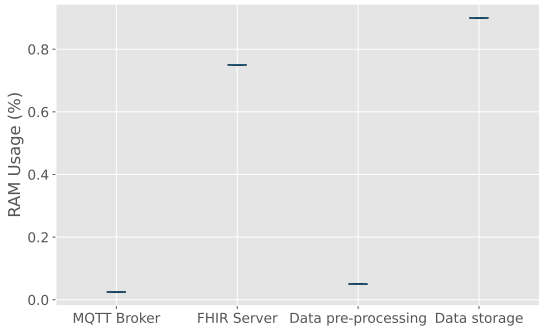


Figure 5.6: RAM usage of each *Smart Gateway* service measured over time during the hospital trial.

When analyzing the data of the trial, the latency values for each *Smart box* were observed to be significantly different from one another. This was later determined to have been caused by clock drift[35], as the service used to maintain the system clock updated (which is the default service that is pre-installed in Ubuntu) could not provide sufficient precision when using public Network Time Protocol (NTP) servers. To solve this, a different strategy is required: the *Smart boxes* must periodically synchronize the system clock directly with the *Smart Gateway*, and the *Smart Gateway* synchronizes its clock with a pool of NTP servers. This minimizes the differences between the system clocks of the *Smart Gateway* and the *Smart boxes*, significantly improving the precision of the timestamps (from ∼100 ms to ∼10 $\mu$s). To implement this, a NTP server has been installed in the *Smart Gateway* as well as a high performance NTP client in both the *Smart Gateway* and *Smart boxes*.

Due to all the aforementioned issues, it is not possible to fully assess the performance of the overall system. As such, additional tests in a controlled environment have been performed to overcome the weaknesses of the initial experiments. These are discussed in the next section.

[35]https://ubuntu.com/server/docs/network-ntp

## 5.2 Laboratory Tests

As mentioned, due to the sparsity of the results obtained in the hospital trial, additional tests in a controlled environment have been performed to fully assess the potential of the system.

The tests have been formulated based on the results of the hospital trial, also using 2 *Smart boxes* and 1 *Smart Gateway*. Since there were BLE connection issues on the *Smart box* reported throughout the hospital trial, for these tests the *Smart boxes* generate simulated data to send via MQTT, *i.e.* we remove the uncertainty derived from the Bluetooth acquisition issues from the *Biostickers* to the *Smart boxes*. In these tests, the focus is the evaluation of MQTT data transmission, thus the FHIR server was not used for these tests. Additionally, to determine if the usage of the custom *Mosquitto* plugin is causing the aforementioned issues, tests have been performed with and without using the authentication plugin. These tests were performed within ISR facilities, so the network infrastructure was already provided by the IT staff. Once again, all devices had static IP addresses to facilitate the deployment and troubleshooting of communication failures.

To assess the performance of the proposed system, the performance metrics defined in the hospital tests were used once more – MQTT bandwidth, CPU and RAM resource usage – as well as:

- MQTT packet loss – Loss of data observed.
- MQTT latency – Interval of time between the data collection at each *Smart box* and the reception of data in the *Smart Gateway*.

### 5.2.1 Results and Discussion

The following graphs have been obtained using the data collected in the laboratory tests using 2 *Smart boxes* and 1 *Smart Gateway*. Figure 5.7 and 5.8 show the measured MQTT bandwidth and latency respectively, averaged over 1 minute.

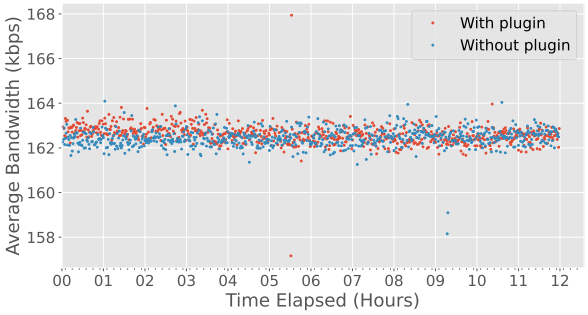


Figure 5.7: Average MQTT bandwidth usage measured over time during the lab tests.

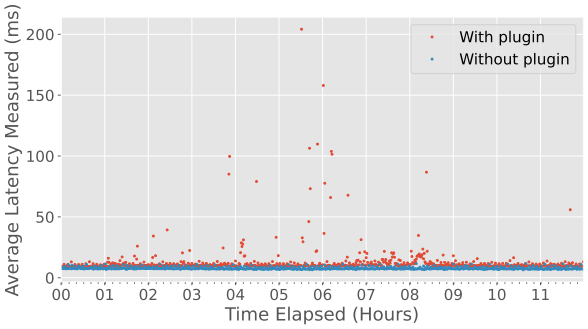


Figure 5.8: Average MQTT latency measured over time during the lab tests.

As seen in the graphs, the MQTT bandwidth is much higher than the value observed in the hospital trials, but still somewhat below the expected value ( 175.68 kbps). This is because the delay caused by the MQTT transmission itself was not accounted for, reducing the transmission rate of the simulated sensors very slightly, which translates into a minor difference in the overall MQTT bandwidth. Nonetheless, it displays much better results with an average bandwidth of $162.6 \pm 0.5$ kbps using the plugin and $162.5 \pm 0.4$ kbps without using the plugin. Despite this final result seemingly favoring the usage of the plugin, it should be noted that the difference between these values is not statistically significant, and thus can be safely disregarded. It means however that the usage of the plugin does not meaningfully impact the performance of the MQTT communications, while also providing authentication and authorization features to the broker.

One thing to note is that the average latency does show a significant increase ($13.22 \pm 14.07$ ms when using the plugin to $7.86 \pm 0.98$ ms without it), which is expected since the plugin adds a fixed processing delay caused by the requests to the data storage service. However, the latency obtained using the plugin seems to be very sparse (as seen with a standard deviation larger than the latency value itself), which would indicate some sort of data loss, but this is not the case. A full analysis revealed that **100%** of the data generated by the *Smart boxes* in this test was captured by the *Smart Gateway*. This means that this delay is being compensated somehow, or that is caused by something else, for example, an issue with the recently introduced NTP server, and so it should be researched further. One thing is clear however, and that is that the data is being successfully transmitted and received using MQTT.

Figure 5.9, 5.10, 5.11 and 5.12 show the measured RAM and CPU usage, with and without the custom authentication plugin. These values are consistent with those observed in the hospital trials. It is observed that the CPU usage of the data storage nearly doubles when using the plugin, which is expected since the plugin performs requests to the data storage whenever a message is received, effectively doubling the amount of requests performed on the data storage – one request on the plugin to check if the *Smart box* is authorized, one request on the data pre-processing service to store the received data.

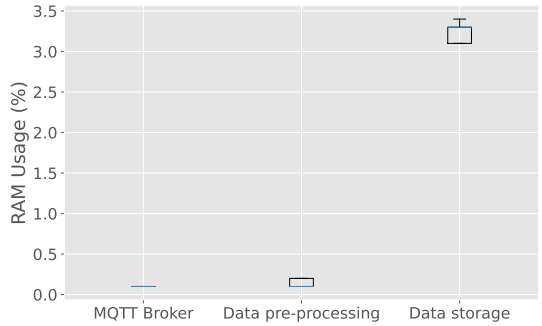


Figure 5.9: Average RAM usage of each *Smart Gateway* service measured over time during the lab tests, when using the custom plugin for Mosquitto.

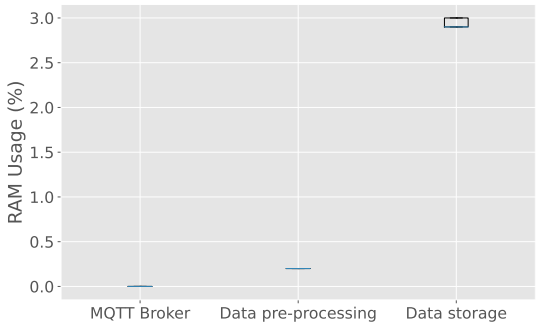


Figure 5.10: Average RAM usage of each *Smart Gateway* service measured over time during the lab tests, without using the custom plugin for Mosquitto.

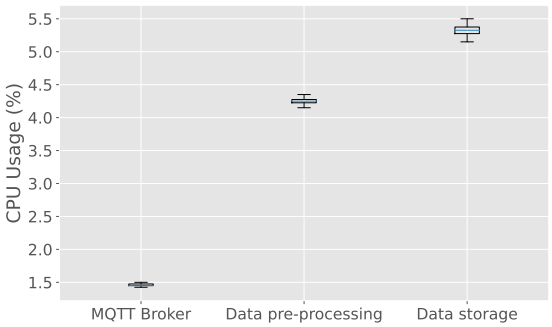


Figure 5.11: Average CPU usage of each *Smart Gateway* service measured over time during the lab tests, when using the custom plugin for Mosquitto.

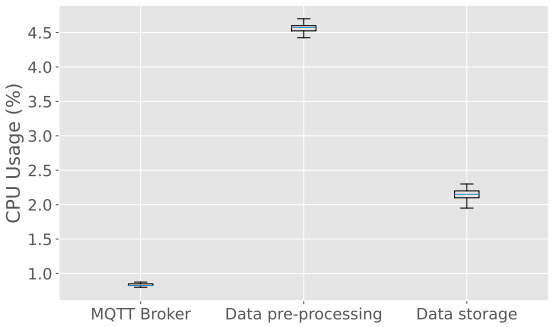


Figure 5.12: Average CPU usage of each *Smart Gateway* service measured over time, without using the custom plugin for Mosquitto.

Overall, the results reported are extremely positive, demonstrating the performance of the system developed throughout this past year, for example, as seen in Figure 5.13, where the sensor measurements sent by the *Smart Gateway* can be observed on the GlobalCare

HIS user interface. However, certain phenomena should be researched further, for example the observed latency deviations on MQTT communications, but nonetheless, the developed architecture should provide a solid foundation for the ongoing development of the WoW project.

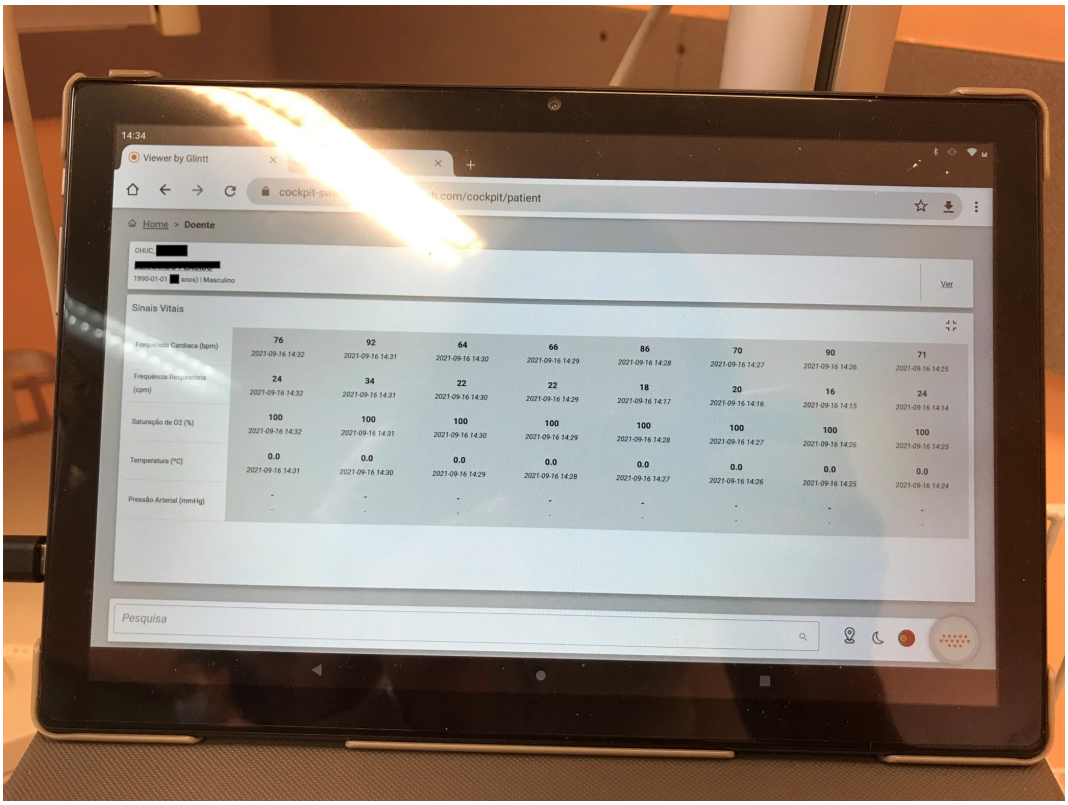


Figure 5.13: Image of the GlobalCare HIS user interface showing measurements sent by the *Smart Gateway*.

## 5.3 Summary

In this chapter, the performance of the proposed system has been experimentally evaluated and discussed. In the next, and final chapter, an overview of the work achieved is presented in light of the proposed contributions, concluding with some final remarks.

# 6 Conclusion

In this dissertation, a new IoT architecture for a pervasive healthcare application of long-term wireless biomonitoring of patients has been proposed and implemented. The work includes an extensive study of BLE communication and a hardware evaluation of different SBCs to support the decision of which IoT hardware platforms would be used in the WoW project. This work also evaluates the performance of two BLE adapters, identifying issues with the BLE stack implementation on *Linux* and their significance to the community. Additionally, a MQTT specification using standardized data formats and security protocols, as well as an extensive service architecture within the *Smart Gateway*, has been proposed, promoting security and interoperability in healthcare IT systems by using an API built with the open standard FHIR to integrate the data in an existing HIS.

Due to external causes, the performance could not be fully evaluated within a trial in clinical facilities, but valuable insight was obtained from it. To overcome the weaknesses of the preliminary tests, results in a controlled lab scenario were extracted for further evaluation. These tests were very promising, showing how the system is more than capable of handling the communications bandwidths required for the project while remaining extremely secure and scalable, thus demonstrating the current capabilities and potential of the proposed solution.

As a result of the developed work and its contributions to the WoW project, an article [55] has been co-written and submitted to the Internet of Things journal published by Elsevier, pending review at the time of writing.

## 6.1 Future Work

To further improve the current work, some open issues of the proposed solutions could be tackled. The proposed MQTT specification does not implement any redundancy mechanisms

exchanging data after interruptions in the communication, *e.g.* network failures, which are required to make the system more robust. Additionally, as mentioned previously, the Data pre-processing service does not implement analytics to detect critical conditions. This would improve significantly the appeal of the system, as it would be capable of providing insight into the patients' conditions in real-time, *e.g.* detecting the eminence of heart attacks or fall events, triggering an immediate alert to healthcare providers. Moreover, the FHIR server currently does not support the creation or modification of subscriptions using the HTTP API as it was not considered for the first trial. However, this should be included in the planning for the final stage of the project.

# Bibliography


[1] K. V. den Heede, N. Bouckaert, and C. V. de Voorde, "The impact of an ageing population on the required hospital capacity: results from forecast analysis on administrative data," vol. 10, no. 5, pp. 697–705, Jul. 2019.

[2] A. Redondi, M. Chirico, L. Borsani, M. Cesana, and M. Tagliasacchi, "An integrated system based on wireless sensor networks for patient monitoring, localization and tracking," *Ad Hoc Networks*, vol. 11, no. 1, pp. 39–53, jan 2013.

[3] European Union, "EU4Health," 2021. [Online]. Available: https://ec.europa.eu/health/

[4] World Health Organization, *Global Strategy on Digital Health*, 2020, vol. 57, no. 4.

[5] J. R. C. Faria, "Estruturação de Sistemas e Aplicações de IIoT em Redes Elétricas Inteligentes," Ph.D. dissertation, University of Coimbra, 2020.

[6] S. Shoja and A. Jalali, "A study of the Internet of Things in the oil and gas industry," *2017 IEEE 4th International Conference on Knowledge-Based Engineering and Innovation, KBEI 2017*, vol. 2018-Janua, pp. 230–236, 2018.

[7] N. Gershenfeld, R. Krikorian, and D. Cohen, "The internet of things," *Scientific American*, vol. 291, no. 4, pp. 76–81, 2004.

[8] A. Darwish and A. E. Hassanien, "Wearable and implantable wireless sensor network solutions for healthcare monitoring," *Sensors*, vol. 11, no. 6, pp. 5561–5595, 2011.

[9] F. Dursun Ergezen and E. Kol, "Nurses' responses to monitor alarms in an intensive care unit: An observational study," *Intensive and Critical Care Nursing*, vol. 59, p. 102845, 2020.

[10] A. F. Silva and M. Tavakoli, "Domiciliary hospitalization through wearable biomonitoring patches: Recent advances, technical challenges, and the relation to covid-19," *Sensors (Switzerland)*, vol. 20, no. 23, pp. 1–35, 2020.

[11] J. Stapleton, *DSDM, dynamic systems development method: the method in practice*. Cambridge University Press, 1997.

[12] G. Aceto, V. Persico, and A. Pescapé, "Industry 4.0 and Health: Internet of Things, Big Data, and Cloud Computing for Healthcare 4.0," *Journal of Industrial Information Integration*, vol. 18, no. February, p. 100129, 2020. [Online]. Available: https://doi.org/10.1016/j.jii.2020.100129

[13] S. B. Baker, W. Xiang, and I. Atkinson, "Internet of Things for Smart Healthcare: Technologies, Challenges, and Opportunities," *IEEE Access*, vol. 5, pp. 26 521–26 544, 2017.

[14] C. Doukas and I. Maglogiannis, "Bringing IoT and Cloud Computing towards Pervasive Healthcare," in *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IEEE, jul 2012, pp. 922–926. [Online]. Available: http://ieeexplore.ieee.org/document/6296978/

[15] T. Wu, F. Wu, C. Qiu, J. M. Redoute, and M. R. Yuce, "A Rigid-Flex Wearable Health Monitoring Sensor Patch for IoT-Connected Healthcare Applications," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6932–6945, 2020.

[16] Yuan Jie Fan, Yue Hong Yin, Li Da Xu, Yan Zeng, and Fan Wu, "IoT-Based Smart Rehabilitation System," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1568–1577, may 2014.

[17] L. Catarinucci, D. de Donno, L. Mainetti, L. Palano, L. Patrono, M. L. Stefanizzi, and L. Tarricone, "An IoT-Aware Architecture for Smart Healthcare Systems," *IEEE Internet of Things Journal*, vol. 2, no. 6, pp. 515–526, dec 2015.

[18] T. Adame, A. Bel, A. Carreras, J. Melià-Seguí, M. Oliver, and R. Pous, "CUIDATS: An RFID–WSN hybrid monitoring system for smart health care environments," *Future Generation Computer Systems*, vol. 78, pp. 602–615, jan 2018.

[19] E. Choi, M. T. Bahadori, A. Schuetz, W. F. Stewart, and J. Sun, "Doctor AI: Predicting Clinical Events via Recurrent Neural Networks." *JMLR workshop and conference proceedings*, vol. 56, pp. 301–318, 2016.

[20] Cisco, "The Internet of Things Reference Model," 2014. [Online]. Available: http://cdn.iotwf.com/resources/71/IoT_Reference_Model_White_Paper_June_4_2014.pdf

[21] F. Wu, T. Wu, and M. R. Yuce, "Design and Implementation of a Wearable Sensor Network System for IoT-Connected Safety and Health Applications," in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. IEEE, apr 2019, pp. 87–90.

[22] L. Minh Dang, M. J. Piran, D. Han, K. Min, and H. Moon, "A survey on internet of things and cloud computing for healthcare," *Electronics (Switzerland)*, vol. 8, no. 7, pp. 1–49, 2019.

[23] P. Gope and T. Hwang, "BSN-Care: A Secure IoT-Based Modern Healthcare System Using Body Sensor Network," *IEEE Sensors Journal*, vol. 16, no. 5, pp. 1368–1376, 2016.

[24] D. Hanes, G. Salgueiro, P. Grossetete, R. Barton, and J. Henry, *IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things*, 1st ed. Cisco Press, 2017.

[25] P. Fuhrer and D. Guinard, "Building a smart hospital using RFID technologies," *European Conference on eHealth 2006, Proceedings of the ECEH 2006*, pp. 131–142, 2006.

[26] EPCglobal, "Specification for RFID Air Interface EPC ™ Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 860 MHz – 960 MHz," *Intellectual Property*, no. October, 2006.

[27] A. Dementyev, S. Hodges, S. Taylor, and J. Smith, "Power Consumption Analysis of Bluetooth Low Energy, ZigBee, and ANT Sensor Nodes in a Cyclic Sleep Scenario," in *Proceedings of IEEE International Wireless Symposium (IWS)*. IEEE, 2013.

[28] J. N. S. Rubí and P. R. L. Gondim, "IoMT platform for pervasive healthcare data aggregation, processing, and sharing based on oneM2M and openEHR," *Sensors (Switzerland)*, vol. 19, no. 19, pp. 1–25, 2019.

[29] RedHat, "Cloud Computing - IaaS vs PaaS vs SaaS." [Online]. Available: https://www.redhat.com/en/topics/cloud-computing/iaas-vs-paas-vs-saas

[30] A. F. Subahi, "Edge-Based IoT Medical Record System: Requirements, Recommendations and Conceptual Design," *IEEE Access*, vol. 7, pp. 94 150–94 159, 2019.

[31] B. Xu, L. D. Xu, H. Cai, C. Xie, J. Hu, and F. Bu, "Ubiquitous data accessing method in iot-based information system for emergency medical services," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1578–1586, 2014.

[32] IBM, "Application Programming Interface (API)." [Online]. Available: https://www.ibm.com/cloud/learn/api

[33] HL7, "FHIR v4.0.1," 2019. [Online]. Available: https://www.hl7.org/fhir/

[34] C. Peng and P. Goswami, "Meaningful integration of data from heterogeneous health services and home environment based on ontology," *Sensors (Switzerland)*, vol. 19, no. 8, 2019.

[35] J. Gruendner, T. Schwachhofer, P. Sippl, N. Wolf, M. Erpenbeck, C. Gulden, L. A. Kapsner, J. Zierk, S. Mate, M. Stürzl, R. Croner, H. U. Prokosch, and D. Toddenroth, "Ketos: Clinical decision support and machine learning as a service – A training and deployment platform based on Docker, OMOP-CDM, and FHIR Web Services," *PLoS ONE*, vol. 14, no. 10, pp. 1–16, 2019.

[36] K. B. Wagholikar, J. C. Mandel, J. G. Klann, N. Wattanasin, M. Mendis, C. G. Chute, K. D. Mandl, and S. N. Murphy, "SMART-on-FHIR implemented over i2b2," *Journal of the American Medical Informatics Association : JAMIA*, vol. 24, no. 2, pp. 398–402, 2017.

[37] A. Raposo, L. Marques, R. Correia, F. Melo, J. Valente, T. Pereira, L. B. Rosário, F. Froes, J. Sanches, and H. P. da Silva, "E-covig: A novel mhealth system for remote monitoring of symptoms in covid-19," *Sensors*, vol. 21, no. 10, pp. 1–18, 2021.

[38] "Transport layer security protocol version 1.3." [Online]. Available: https://tools.ietf.org/html/rfc8446

[39] Personal Connected Health Alliance, "Continua Adoption Playbook Deploying Interoperable Connected Health in Your Health System," no. May, p. 20, 2017. [Online]. Available: https://www.pchalliance.org/continua-adoption-playbook

[40] R. Jain, "Introduction to raspberry pi," in *Advanced Home Automation Using Raspberry Pi*. Springer, 2021, pp. 1–22.

[41] J. C. Pierce, "The fibonacci series," *The Scientific Monthly*, vol. 73, no. 4, pp. 224–228, 1951.

[42] H. J. Jeffrey, "Chaos game visualization of sequences," *Computers and Graphics*, vol. 16, no. 1, pp. 25–33, 1992.

[43] D. P. Playne, M. G. B. Johnson, and K. A. Hawick, "Benchmarking GPU Devices with N-Body Simulations," *Proc. 2009 International Conference on Computer Design (CDES 09) July, Las Vegas, USA.*, pp. 150–156, 2009.

[44] B. Specification, "Bluetooth ®Specification Bluetooth Core Specification," 1999. [Online]. Available: https://www.bluetooth.com/specifications/adopted-specifications

[45] Z. K. Farej and A. M. Saeed, "Analysis and Performance Evaluation of Bluetooth Low Energy Piconet Network," *OALib*, vol. 07, no. 10, pp. 1–11, 2020.

[46] "Bluez." [Online]. Available: https://www.bluez.org/

[47] C. Gomez, J. Oller, and J. Paradells, "Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology," *Sensors*, vol. 12, no. 9, pp. 11 734–11 753, 2012. [Online]. Available: https://www.mdpi.com/1424-8220/12/9/11734

[48] K. Wright and H. Kang, "Performance analysis of various mechanisms for inter-process communication," *Operating Systems and Networks Lab, Dept. of . . .*, 2007.

[49] C. Asiminidis, G. Kokkonis, and S. Kontogiannis, "Database systems performance evaluation for IoT applications," 2018. [Online]. Available: https://doi.org/10.2139/ssrn.3360886

[50] "Db-engines monthly popularity ranking." [Online]. Available: https://db-engines.com/en/ranking

[51] "Hl7 fhir r4 specification," accessed in 10-December-2021. [Online]. Available: https://www.hl7.org/fhir/

[52] J. Clarke, *SQL injection attacks and defense*. Waltham, Mass: Elsevier, 2012.

[53] "Eclipse mosquitto - an open source mqtt broker," accessed in 10-December-2021. [Online]. Available: https://mosquitto.org/documentation/

[54] "Hapi fhir - java api for hl7 fhir clients and servers," accessed in 10-December-2021. [Online]. Available: https://hapifhir.io/

[55] F. Famá, J. Faria, D. Portugal, "An IoT-based Interoperable Architecture for Wireless Patient Biomonitoring and Digital Healthcare," *Internet of Things, Elsevier*, 2021, (Under Review).

# Appendix A

# MQTT Payload Formats

In the implemented system, the field *message_type* defines what type of message is communicated, and must be one of the following:

- "MEASUREMENT_TEMPERATURE": The payload format for this message is:

```
"payload": {
    "temperature" : 10.0,
    "is_celsius" : true
}

```

where the "temperature" field is the temperature measurement, and "is_celsius" field indicates whether the measurement value it is in Celsius or Fahrenheit.

- "MEASUREMENT_IMU": The payload format for this message is:

```
"payload": {
    "imu": {
        "linear_acceleration": {"x": 0.00, "y": 0.00, "z": 0
    ↪ .00},
        "angular_velocity": {"x": 0.00, "y": 0.00, "z": 0.00}
    }
        "pose_description" : "SITTING"
}

```

where the "linear_acceleration" field is the accelerometer measurement,"angular_velocity" field is the gyroscope measurement, and "pose_description" is the text description of

the current body pose of the patient.

- “MEASUREMENT_ECG”: The payload format for this message is:

```
"payload": {
    "ecg" : 10
}

```

where the “ecg” field is the ECG measurement.

- “MEASUREMENT_PULSEOXIMETRY”: The payload format for this message is:

```
"payload": {
    "spo2" : 10.0
}

```

where the “spo2” field is the pulse oximetry measurement.

- “MEASUREMENT_HR”: The payload format for this message is:

```
"payload": {
    "bpm" : 10.0
}

```

where the “bpm” field is the heart rate measurement.

- “MEASUREMENT_RESPIRATION”: The payload format for this message is:

```
"payload": {
    "respiration" : 10.0
}

```

where the “respiration” field is the respiration rate measurement.

# Appendix B

# FHIR Resource JSON Representations

In the implemented system, different FHIR Resources are used to exchange information with the HIS using JSON representations. Some examples are shown below:

- FHIR resource for the *Smart box*:

```
{
    "resourceType": "Device",
    "identifier": [
        {
            "system": "urn:ietf:rfc:3986",
            "value":
    ↪ "urn:uuid:61ebe359-bfdc-4613-8bf2-c5e300945f0a"
        }
    ],
    "type": {
        "coding": [
            {
                "system": "http://snomed.info/sct",
                "code": "5159002",
                "display": "Physiologic monitoring system"
            }
        ],
        "text": "Smartbox"
    }
}

```

- FHIR resource for a temperature sensor, for other sensors a different code must be used in the “code” entry:

```
{
    "resourceType": "Device",
    "identifier": [
        {
            "system": "urn:ietf:rfc:3986",
            "value":
↪ "urn:uuid:88f151c0-a954-468a-88bd-5ae15c08e059"
        }
    ],
    "type": {
        "coding": [
            {
                "system": "http://snomed.info/sct",
                "code": "27991004",
                "display": "Thermometer"
            }
        ],
        "text": "Thermometer"
    },
    "parent": {
        "reference":
↪ "urn:uuid:61ebe359-bfdc-4613-8bf2-c5e300945f0a"
    }
}

```

- FHIR resource for temperature measurement, for different measurements a different code must be used in the “code” entry:

```
{
    "resourceType": "Observation",
    "status": "final",
    "category": [
        {
```

```
            "coding": [
                {
                    "system":
↪ "http://terminology.hl7.org/CodeSystem/observation-category",
                    "code": "vital-signs",
                    "display": "Vital Signs"
                }
            ]
        }
    ],
    "code": {
        "coding": [
            {
                "system": "http://loinc.org",
                "code": "8310-5",
                "display": "Body temperature"
            }
        ]
    },
    "bodySite": {
        "coding": [
            {
                "system": "http://snomed.info/sct",
                "code": "74262004",
                "display": "Oral cavity"
            }
        ],
        "text": "Oral cavity"
    },
    "effectiveInstant": "2017-01-01T00:00:00.000Z",
    "valueQuantity": {
        "value": 38,
        "unit": "C",
        "system": "http://unitsofmeasure.org",
        "code": "Cel"
    },
```

```
    "device": {
        "reference":
    ↪ "urn:uuid:88f151c0-a954-468a-88bd-5ae15c08e059"
    }
}

```

- FHIR resource for communicating temperature measurement, which is a bundle of the temperature measurement resource, the temperature sensor resource and the *Smart box* resource:

```
{
    "resourceType": "Bundle",
    "type": "transaction",
    "entry": [
        {
            "fullUrl":
    ↪ "urn:uuid:ca7a77cc-f1c4-4227-9a2a-b73833bfbb11",
            "resource": {
                "resourceType": "Observation",
                "status": "final",
                "category": [
                    {
                        "coding": [
                            {
                                "system":
    ↪ "http://terminology.hl7.org/CodeSystem/observation-category",
                                "code": "vital-signs",
                                "display": "Vital Signs"
                            }
                        ]
                    }
                ],
                "code": {
                    "coding": [
                        {
```

```
                            "system": "http://loinc.org",
                            "code": "8310-5",
                            "display": "Body temperature"
                        }
                    ]
                },
                "bodySite": {
                    "coding": [
                        {
                            "system": "http://snomed.info/sct",
                            "code": "74262004",
                            "display": "Oral cavity"
                        }
                    ],
                    "text": "Oral cavity"
                },
                "effectiveInstant": "2017-01-01T00:00:00.000Z",
                "valueQuantity": {
                    "value": 38,
                    "unit": "C",
                    "system": "http://unitsofmeasure.org",
                    "code": "Cel"
                },
                "device": {
                    "reference":
    ↪ "urn:uuid:88f151c0-a954-468a-88bd-5ae15c08e059"
                }
            },
            "request": {
                "method": "POST",
                "url": "Observation"
            }
        },
        {
            "fullUrl":
    ↪ "urn:uuid:88f151c0-a954-468a-88bd-5ae15c08e059",
```

```
            "resource": {
                "resourceType": "Device",
                "identifier": [
                    {
                        "system": "urn:ietf:rfc:3986",
                        "value":
    ↪ "urn:uuid:88f151c0-a954-468a-88bd-5ae15c08e059"
                    }
                ],
                "type": {
                    "coding": [
                        {
                            "system": "http://snomed.info/sct",
                            "code": "27991004",
                            "display": "Thermometer"
                        }
                    ],
                    "text": "Thermometer"
                },
                "parent": {
                    "reference":
    ↪ "urn:uuid:61ebe359-bfdc-4613-8bf2-c5e300945f0a"
                }
            },
            "request": {
                "method": "POST",
                "url": "Device"
            }
        },
        {
            "fullUrl":
    ↪ "urn:uuid:61ebe359-bfdc-4613-8bf2-c5e300945f0a",
            "resource": {
                "resourceType": "Device",
                "identifier": [
                    {
```

```
                        "system": "urn:ietf:rfc:3986",
                        "value":
    ↪ "urn:uuid:61ebe359-bfdc-4613-8bf2-c5e300945f0a"
                    }
                ],
                "type": {
                    "coding": [
                        {
                            "system": "http://snomed.info/sct",
                            "code": "5159002",
                            "display": "Physiologic monitoring
    ↪ system"
                        }
                    ],
                    "text": "Smartbox"
                }
            },
            "request": {
                "method": "POST",
                "url": "Device"
            }
        }
    ]
}

```