Smart Gateway
System range
Up to 5 meters.
Glintt
GlobalCare
Bio-monitoring
software.

José Nuno da Cruz Faria

Wireless IoT Smart Bed System

October 2021

Universidade de Coimbra

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Wireless IoT Smart Bed System

José Nuno da Cruz Faria

Coimbra, October 2021

· U C ·

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, October 2021

Acknowledgments

To-do: Add acknowledgements text.

Resumo

To-do: Add abstract text.

Abstract

To-do: Add abstract text.

Contents

List of Acronyms

To-do: Remember to check the order on the acronyms!!

API	Application Programming Interface
BLE	Bluetooth Low Energy
CoAP	Constrained Application Protocol
ECG	Electrocardiogram
EHR	Eletronic Health Record
EPC/RFID	EPCglobal Gen2 RFID
FHIR	Fast Healthcare Interoperability Resources
HIS	Health Information System
IMU	Inertial Measurement Unit
ISR	Institute of Systems and Robotics
IoT	Internet of Things
IP	Internet Protocol
IT	Information Technology
MQTT	Message Queuing Telemetry Transport
MTU	Maximum Transmission Unit
OSI	Open System Interconnection
PPG	Photoplethysmography

RDBMS	Relational Database Management System
RFID	Radio-frequency Identification
SBC	Single Board Computer
SQL	Structured Query Language
TLS	Transport Layer Security
UI	User Interface
PHY	Physical Layer
ATT	Attribute Protocol
GATT	General Attribute Profile
WBAN	Wireless Body Area Network
JSON	JavaScript Object Notation
WoW	Wireless biOmonitoring stickers and smart bed architecture: toWards Untethered Patients
GUI	Graphical User Interface
PDU	Protocol Data Unit
SMP	Security Manager
ATT	Attribute Protocol
GATT	Generic Attribute Protocol
GAP	Generic Access Profile
DLE	Data Length Extension
LL	Link Layer
SoC	System on a Chip
L2CAP	Logical Link Control and Adaptation Protocol

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 - both human resources and medical equipment, hospital beds, etc[1, 2]. 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 the 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 to "physical objects" or groups of these, allowing them to be capable of connecting with other devices and exchange data through the Internet [7]. Today, we find that 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 them much-needed comfort (perhaps even allowing them to return to their own homes), 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 [8]. 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) 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 [9].

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 tackling crucial issues like those described previously.

The requirements for the system, and their priorities based on a MoSCoW prioritization analysis [10], 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 developed. 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 *SmartBox*. 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, we reflect upon the work developed and discuss the completion of the 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, we present a comparative analysis of similar works in the literature, identifying their strengths and weaknesses. 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 [11].

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 [12]:

To-do: Find alternatives for ’strategy’ - communication strategy, architecture, technology? I opted with technology, awaiting feedback

> IoT is a technology supported on 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 [13, 14, 15]. 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 [16, 17]. 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 [18]. 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 [19] 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 we will focus on pervasive healthcare applications.

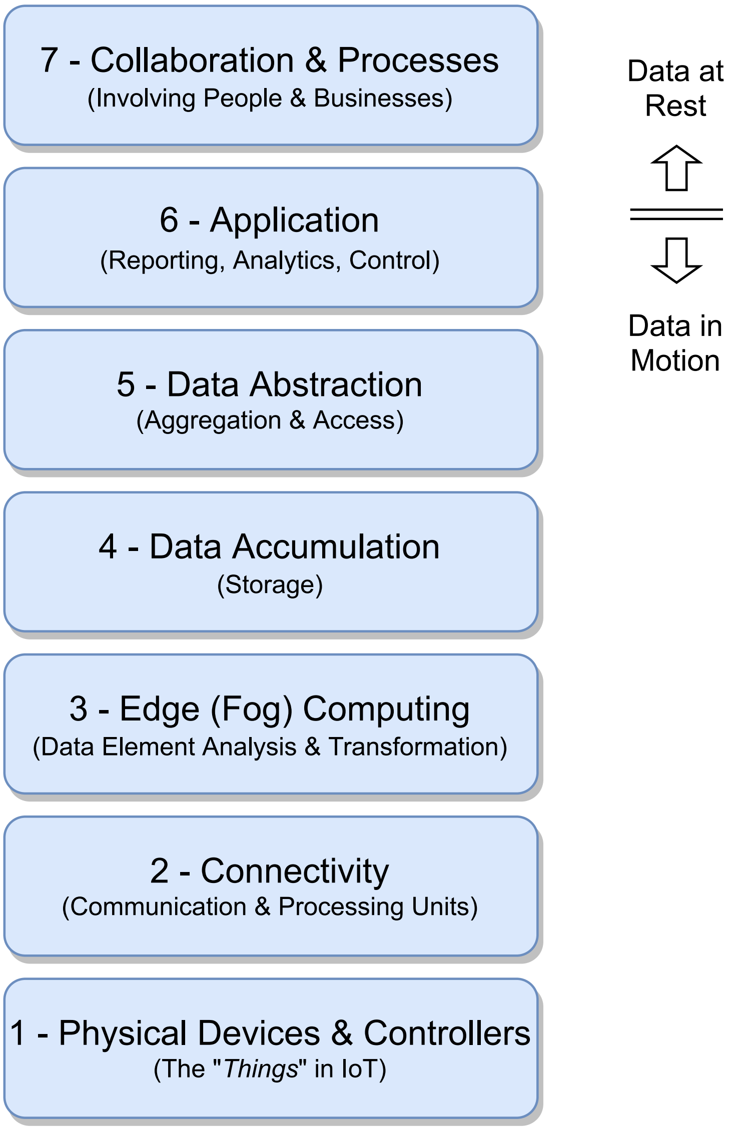

7 - Collaboration & Processes
(Involving People & Businesses)
6 - Application
(Reporting, Analytics, Control)
5 - Data Abstraction
(Aggregation & Access)
4 - Data Accumulation
(Storage)
3 - Edge (Fog) Computing
(Data Element Analysis & Transformation)
2 - Connectivity
(Communication & Processing Units)
1 - Physical Devices & Controllers
(The "*Things*" in IoT)
Data at Rest
Data in Motion

Figure 2.1: IoT reference model published by IoTWF. Source: [19].

From a hardware perspective, in this work we focus on the most common approach taken by researchers, using 3 distinct components:

- **Endpoint** or **edge** nodes (corresponding to Layer 1), which interact with the physical world, capturing data.
- **Gateway** devices (Layer 3), which connect to one or multiple **edge** nodes, filtering and aggregating the data generated by these and communicating it to a central server;
- **Central** server (Layers 4-6), which is responsible for collecting, storing and analyzing the captured data in order to provide users with valuable insight;

2.2.1 Layer 1: Physical Devices and Controllers

The first layer of the model [19] 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 [12]. 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 [17, 16]. 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. [14]. From the literature [14, 20, 17, 21], 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 [20].

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

Table 2.1: Type of sensors commonly used in pervasive healthcare applications. Adapted from [21].

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 [19]. 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 [12, 16, 17]:

- **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 [16]. 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 [22].

- **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 [23].

Regarding communications within the local network, these generally have short transmission ranges [12]. 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 [13, 20, 14]. ZigBee and Radio-frequency Identification (RFID) are also used in many systems in this domain, particularly in asset tracking oriented applications [24, 16, 17]. Despite the absence of a universal specification for RFID, the most widely used standard is the EPCglobal Gen2 RFID [25]. For the sake of simplicity, we use EPC/RFID to designate it.

Out of the abovementioned technologies, BLE offers greater throughput, better security, and nearly the lowest energy consumption [26]. Table 2.2 shows a comparison between the different protocols.

	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	<100m	<10m, (100m for Active tags)	20m
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

Table 2.2: Overview of the most common communication protocols used within short range. Adapted from [12].

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 [17, 24, 14, 16].

Application Protocols

So far we have discussed the underlying networking technologies that link the devices in system. 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 composed 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.

	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

Table 2.3: Comparison between CoAP and MQTT protocols. Adapted from [23].

In [27], 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.

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.

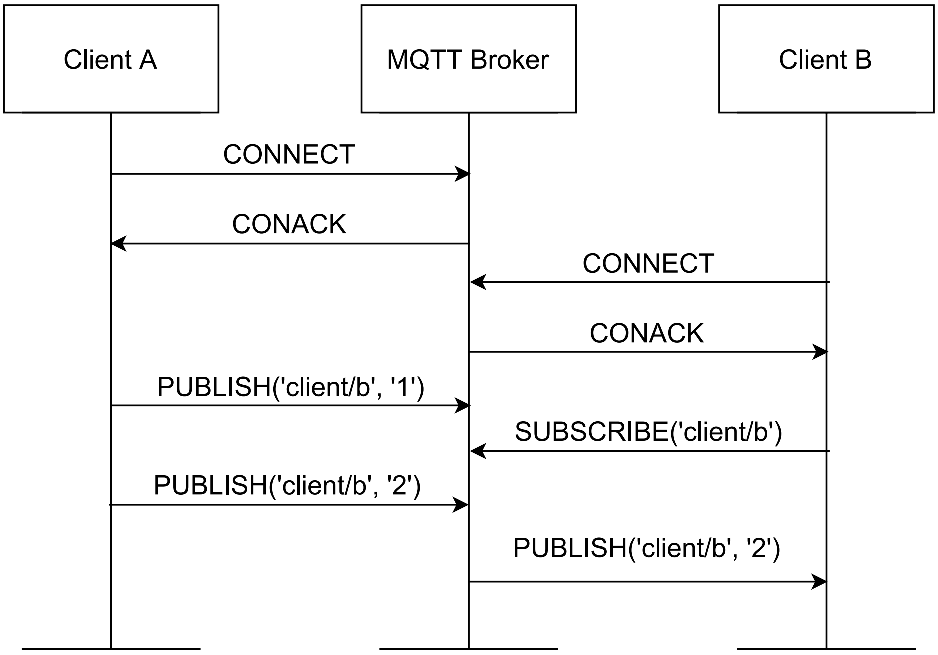


Figure 2.2: Diagram of a MQTT message sequence. In this diagram, we have two clients (A and B) connecting to the MQTT broker, where client A transmits two messages to the topic "client/b" while client B subscribes to that topic to demonstrate how the Publish-Subscribe model works. In MQTT, a client recieves a message whenever a client (including itself) publishes a message on that topic.

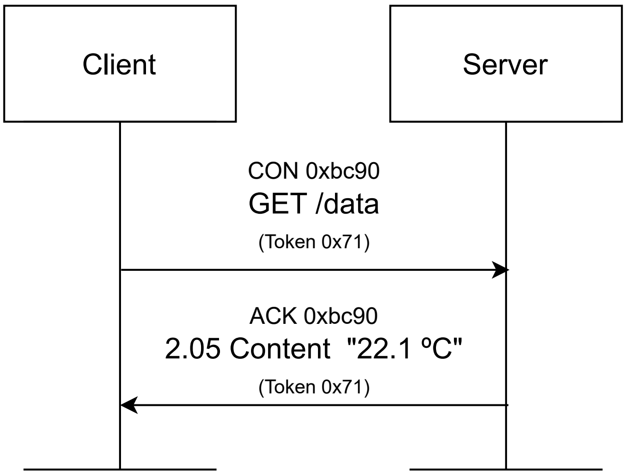


Figure 2.3: Diagram of a CoAP message sequence. In this diagram, we have a client making a GET request to the CoAP server to retrieve information, in order to demonstrate how the Request-Response model works. In CoAP, the requests are delivered asyncronously, and so the response messages must contain a “token” value so that the client can identify which request resulted in this response.

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 [23]. In the Data Accumulation layer, we need to define how the system captures the data and stores it, so it becomes usable for applications 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 [13], 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. We can identify three distinct types of cloud services:

- **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.

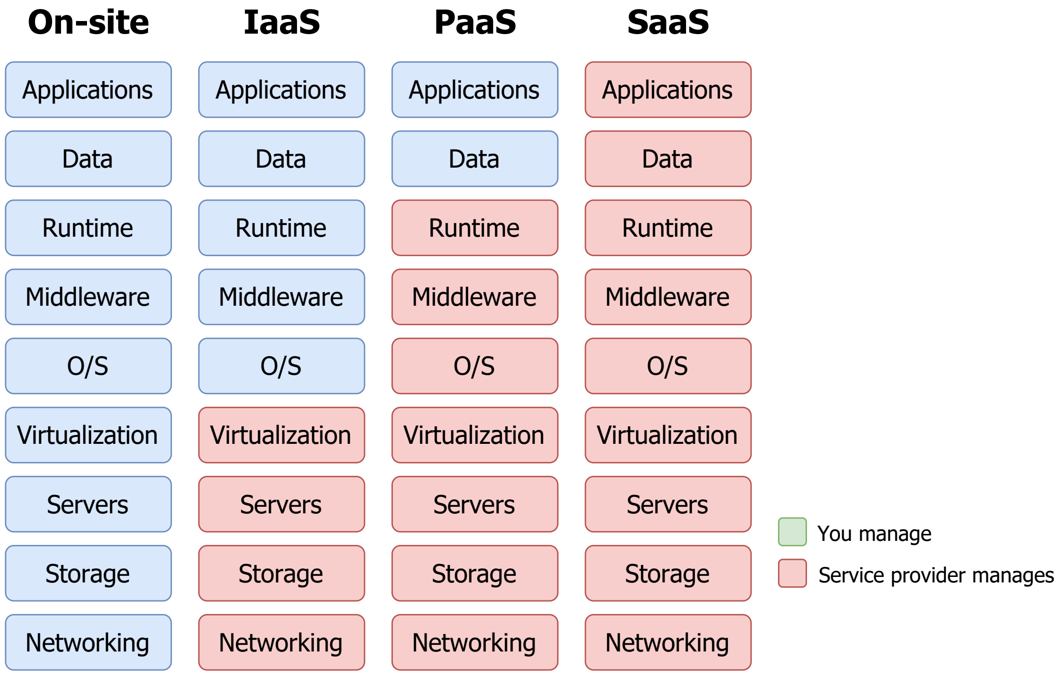


Figure 2.4: Differences between the cloud offerings and on-premise solutions. Source: [28]

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 [12]. 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 [24, 14, 16, 17]. 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 [29] 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 [30]. Table 2.4 illustrates the differences between these two 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

Table 2.4: Comparison between SQL and NoSQL database technologies.

2.2.5 Layer 5: Data Abstraction

In the previous layer, we have defined 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 develop services that simplify how the applications access the data, to reconcile the different data stores and ensure the information is complete and consistent [19]. 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 [31]. 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 [32], which are called Eletronic Health Records (EHRs). EHRs are the digital equivalent of a patient's paper-chart, they contain 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 [33]. 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 [34].

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 User Interfaces (UIs), which provide different functionalities depending on the application. Some may show simple reports regarding the collected data [13, 14], 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 [34, 35, 36].

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 [19]. 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

We have thoroughly discussed how IoT systems are designed, but so far we have not discussed details of any specific implementation so far. This section presents an overview of IoT connected healthcare applications described in the literature, highlighting each of their strengths and weaknesses.

In [24], 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 [17], 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. [14] 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 [13], 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. [36] developed a system called “e-CoVig” a low-cost solution for monitoring COVID-19 patients during the quarantine, as shown in Figure 2.5. 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.

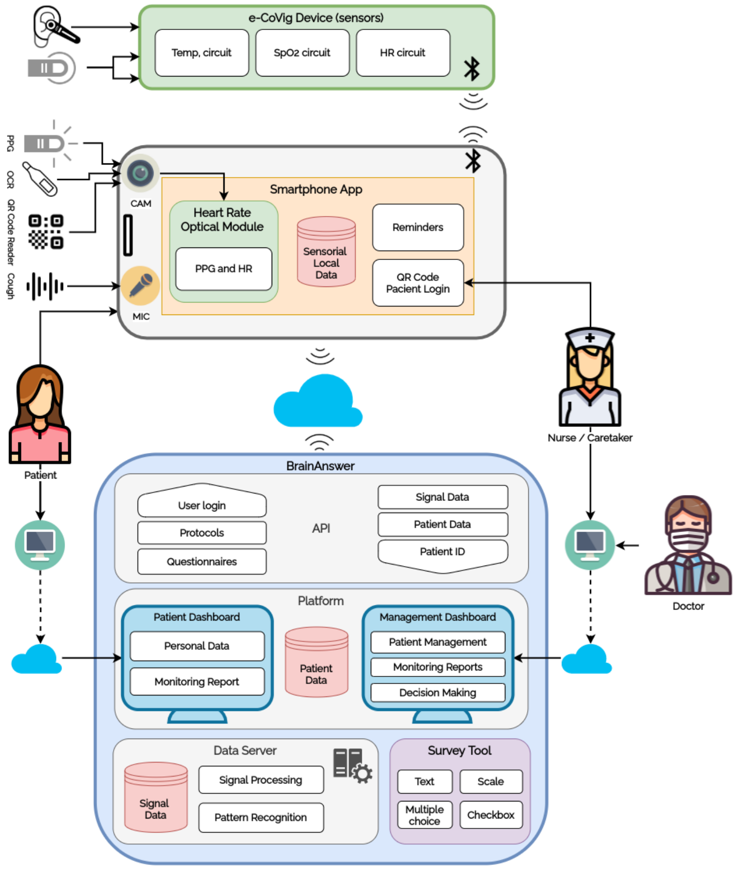


Figure 2.5: Overview of e-Covig’s system architecture. Source: [36].

In order to summarize the key properties of the previously discussed solutions, the following table, Table 2.5 is presented.

References	Measured Signals	Networking Protocols	Data Storage	e-Health Standards	Application Features	Security Features
Fuhrer et al. [24]	N/A	EPC/RFID, Wi-Fi	MySQL	None	RTLS	Unspecified Storage Encryption
Adame et al. [17]	Temperature, Heart Rate, Accelerometer	EPC/RFID, Wi-Fi	MySQL	None	RTLS, Fall Detection, Vital signs monitoring	AES-128, WPA-Personal
Wu et al. [14]	Temperature, Heart Rate, Accelerometer	BLE, Wi-Fi, MQTT	MySQL	None	RTLS, Fall Detection, Vital signs monitoring	AES-128
Doukas et al. [13]	Temperature, Heart Rate, Accelerometer, CO_2 Sensor	BLE, Wi-Fi, GPRS/3G HTTP	MySQL	None	Fall Detection, Vital signs monitoring	AES
Raposo et al. [36]	Temperature, Heart Rate, Pulse Oximetry, Respiration Rate	BLE Wi-Fi	Unknown	None	Fall Detection, Vital signs monitoring, Clinical decision support	Unknown

Table 2.5: Comparison between different pervasive healthcare applications.

2.3.1 Weaknesses of literature

From the literature, we find that many solutions secure communications between the devices using standard encryption algorithms like Advanced Encryption Standard (AES) [17, 14, 13]. However, very few discuss authentication and authorization processes [13, 22]. 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)[1]. 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 [27].

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 [37]. 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.

[1] The Transport Layer Security (TLS) Protocol Version 1.3: https://tools.ietf.org/html/rfc8446

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 as theater networking stack, in an effort of creating battery-less 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, we propose a novel fully modular IoT infrastructure that uses 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 2.6:

- the *Biostickers*, that acquire the patient’s physiological signals;
- the *SmartBox*, which acts as a central node of the WBAN and aggregates the data, communicating it to the gateway;
- 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 *SmartBox* is coupled with an hospital bed, which is often referred internally as a *SmartBed*.

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

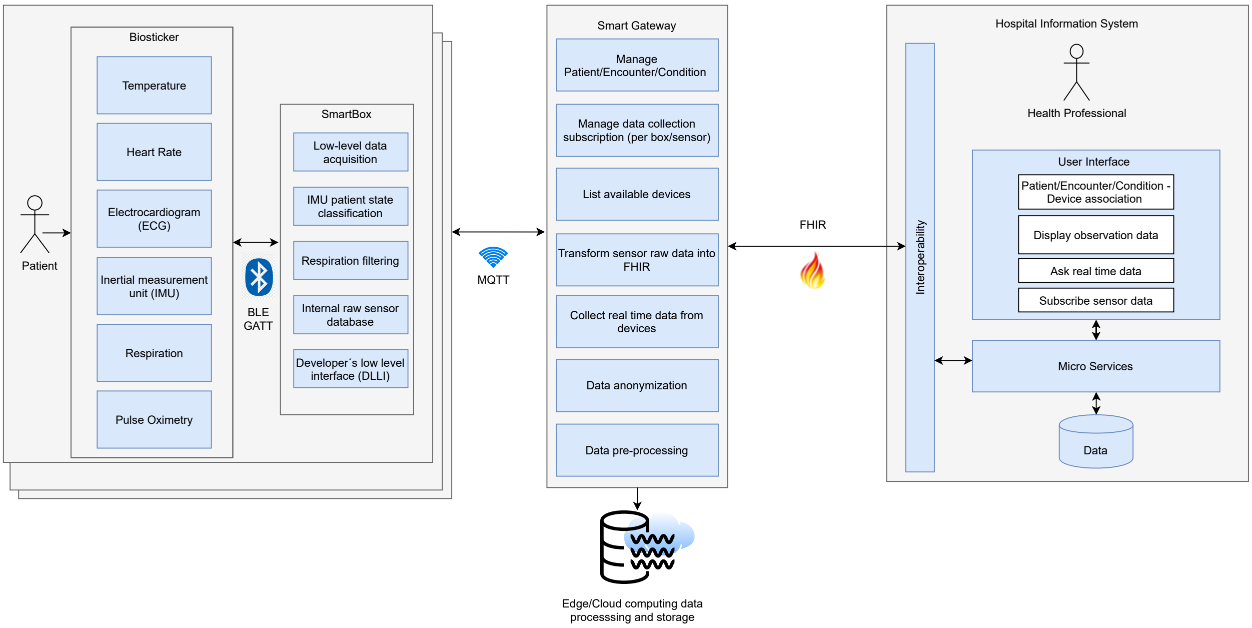

Biosticker
Temperature
Heart Rate
Electrocardiogram (ECG)
Inertial measurement unit (IMU)
Respiration
Pulse Oximetry
Patient
BLE GATT
SmartBox
Low-level data acquisition
IMU patient state classification
Respiration filtering
Internal raw sensor database
Developer´s low level interface (DLLI)
MQTT
Smart Gateway
Manage Patient/Encounter/Condition
Manage data collection subscription (per box/sensor)
List available devices
Transform sensor raw data into FHIR
Collect real time data from devices
Data anonymization
Data pre-processing
Edge/Cloud computing data processsing and storage
FHIR
Hospital Information System
Health Professional
Interoperability
User Interface
Patient/Encounter/Condition - Device association
Display observation data
Ask real time data
Subscribe sensor data
Micro Services
Data

Figure 2.6: System architecture of the WoW project. In this work, we discuss the implementation of the *Smart Gateway* components, as well as contributions to the *SmartBox* development.

For the work developed throughout the dissertation, we propose the following contributions:

- Development and deployment of the *SmartBoxes* 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 development of data integration pipelines using MQTT and management of the multiple SmartBoxes in the Smart Gateway;
- Implementation of a FHIR API layer to integrate the proposed system in the Global-Care HIS.
- Evaluation of the performance of the proposed system on hospital trials within the WoW project.

2.5 Summary

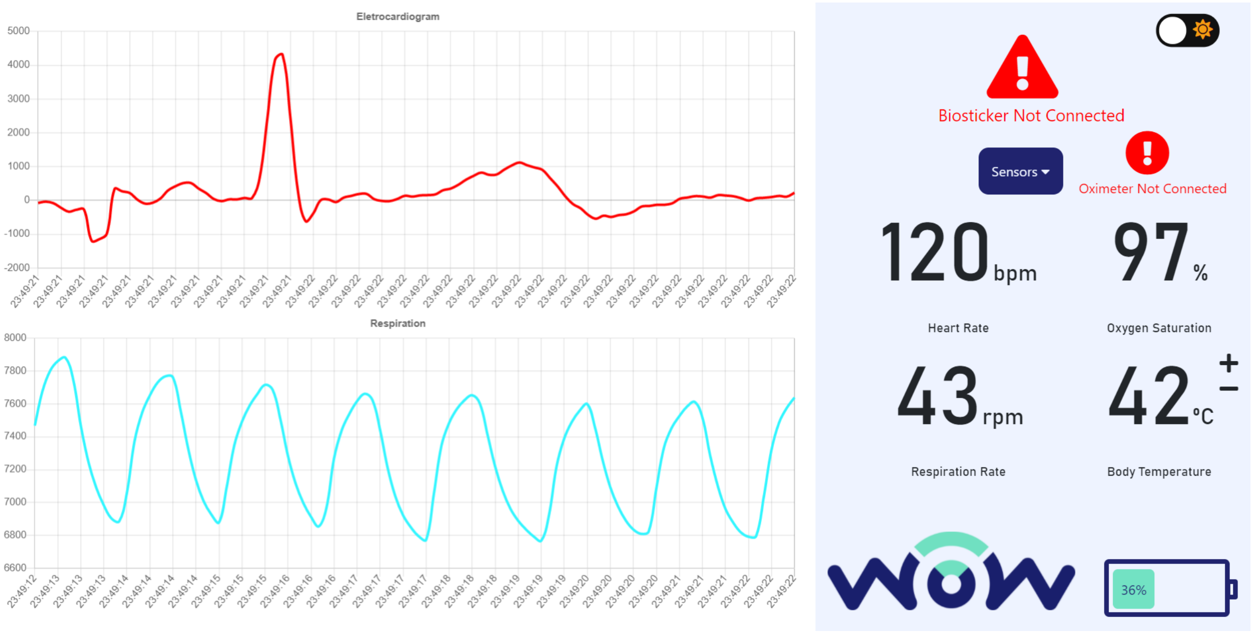
In this chapter, we have discussed the importance of IoT systems, how these are composed and how these can bring value to healthcare providers. After analyzing different relevant systems proposed previously in the literature, we have defined a set of criteria to guide the development of our own implementation.

In the next chapter, we begin with the evaluation of the different IoT kits for the *Smart Bed* hardware, and afterwards, we evaluate different BLE adapters for patient data acquisition to implement secure communications between the *Biostickers* and the *SmartBoxes*.

3 SmartBox Development

In the proposed architecture, the *SmartBox* plays the role of acquiring the data that is transmitted wirelessly by the *Biostickers*. Each *SmartBox* 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 process 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 *SmartBox* to determine the body pose of the patient, as well as filtering the respiration data to account for signal fluctuations caused by sudden movements by the patient.

Additionally, for debugging purposes, researchers at the ISR developed a simple developer GUI, which can be seen on Figure 3.1.



Biosticker Not Connected

Oximeter Not Connected

120 bpm
Heart Rate

97 %
Oxygen Saturation

43 rpm
Respiration Rate

42 °C
Body Temperature

Figure 3.1: Illustration of the developer Graphical User Interface (GUI) for debugging the *SmartBox* acquisition, designed by the WoW research team at ISR.

Regarding data acquisition, the *SmartBox* collects 5 distinct biosignals, which can be seen on the developer low level GUI on Figure 3.1:

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

Todo: Confirm with @Bernardo acquisition rates for each value.

Todo: Ask @Miguel for a picture of the biostickers.

3.1 Deciding on a Hardware Platform

In the context of this dissertation, two different Single Board Computers (SBC) were considered for the development of the *SmartBox*: a Raspberry Pi 4 Model B and an UDOO BOLT v3. In the following sections we discuss and compare the characteristics of each platform.

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 [38]. At the time of the writing, the Raspberry Pi 4 Model B (or Raspberry Pi 4B)[2], which can be seen in Figure 3.2, is the latest revision of the Raspberry Pi series, powered by Broadcom BCM2711 System on a Chip (SoC).

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

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

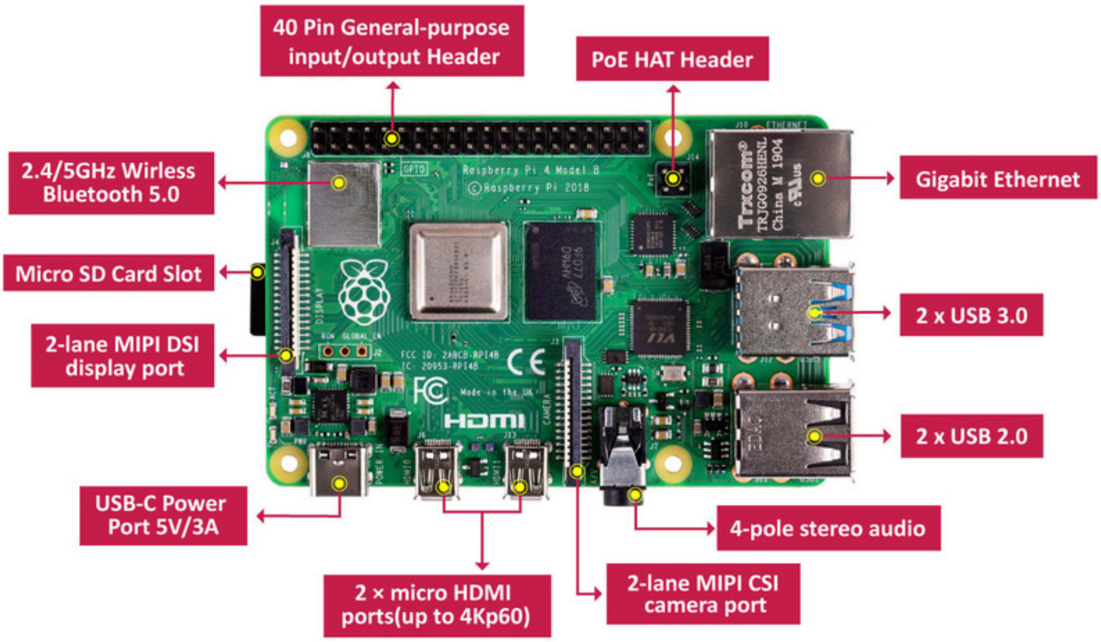

40 Pin General-purpose input/output Header
PoE HAT Header
2.4/5GHz Wirless Bluetooth 5.0
Gigabit Ethernet
Micro SD Card Slot
2 x USB 3.0
2-lane MIPI DSI display port
2 x USB 2.0
USB-C Power Port 5V/3A
4-pole stereo audio
2 × micro HDMI ports(up to 4Kp60)
2-lane MIPI CSI camera port

Figure 3.2: Raspberry Pi 4B.

UDOO BOLT V3

As stated by the manufacturer[3], 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. The UDOO Bolt is incredibly well-supported by UDOO but unfortunately, it does not have nearly the same community support of Raspberry Pi.

The UDOO BOLT V3, which can be seen in Figure 3.3, 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.

To-do: If time allows it, remake this with better quality.

[3] Product Website: https://www.udoo.org/discover-the-udoo-bolt/

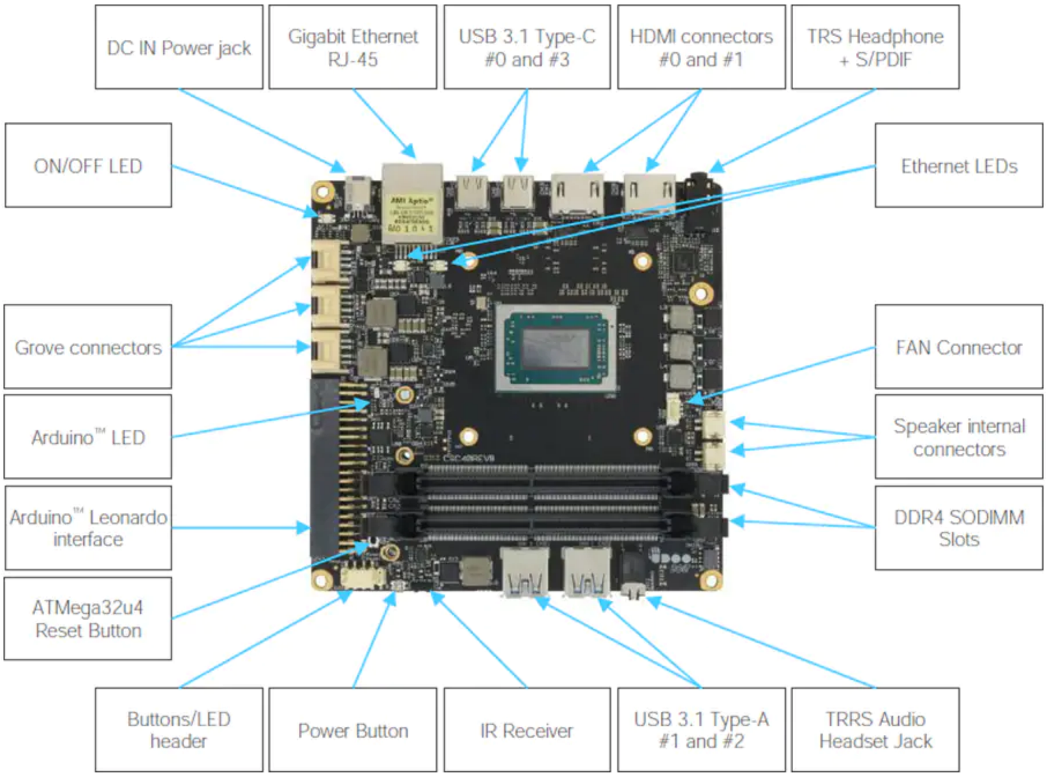


Figure 3.3: UDOO BOLT V3.

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, we immediately conclude that the Raspberry Pi is a much more affordable alternative. At over 1/7 of the price, it already has a working WiFi+BT networking module (which is not included in the UDOO BOLT), 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 delivery a much better overall computing performance.

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

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

In order to understand how these differences in the hardware specification between the SBCs translate to real-world performance, a test suite was developed and conducted to quantify the performance of each SBC. The tools developed for each test can be found here[4].

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

Below, we detail how each test works and discuss how each SBC performed.

Test 1: Python Benchmark

Given the data processing requirements for the *SmartBox*, and as Python will be used as the main scripting language for most of the *SmartBox* development, we developed a simple test 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 [39], and we measure the time taken. This process is repeated 10 times for different numbers, from 10000 to 500000, to determine average run time.

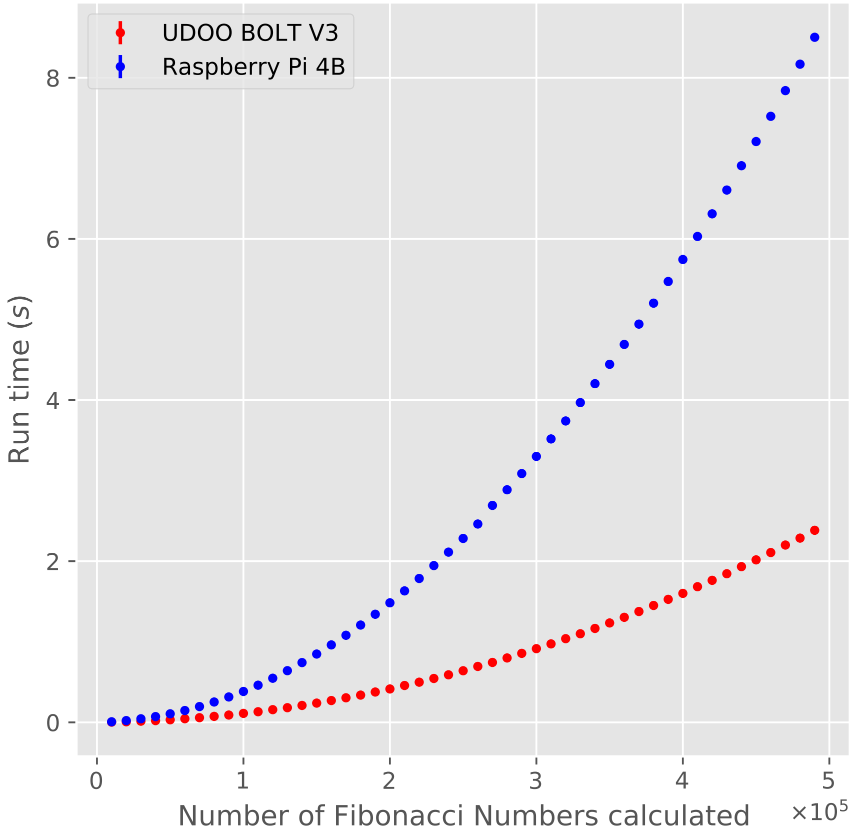


Figure 3.4: Custom Python benchmark for the Raspberry Pi 4B and UDOO BOLT V3. The standard deviation for each run time is inferior to 0.1% in each test, and therefore it is not displayed in the figure.

From Figure 3.4, we can observe that the time taken for computing each number increases exponentially for each platform. By analyzing the data, we can see that the UDOO BOLT V3 outperforms Raspberry Pi 4B on average by a factor of 2.5 ± 0.2.

Test 2: Phoronix Test Suite

The Phoronix Test Suite[5] 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, we chose the following standard test profiles provided by Phoronix[6], which are a compilation of the most popular Python and CPU benchmarks used [7]:

- BYTE Unix Benchmark ("BYTE"), single-threaded CPU benchmark – Runs BYTE UNIX benchmark suite (more particularly, 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 [40] and measures its run-time.

[5] Phoronix Test Suite - Linux Testing & Benchmarking Platform, Automated Testing, Open-Source Benchmarking: `https://www.phoronix-test-suite.com/`

[6] OpenBenchmarking.org - Cross-Platform, Open-Source Automated Benchmarking Platform: `https://openbenchmarking.org/`

[7] The benchmark results obtained were published in the *OpenBenchmarking.org* website, and can be found by visiting the following webpage: `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 [41] and measures its run-time.

- PyPerformance *json_loads* Benchmark ("json"), single-threaded Python & CPU benchmark – Evaluates JavaScript Object Notation (JSON)[8] parsing and serialization, a widely used open stardard 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.

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

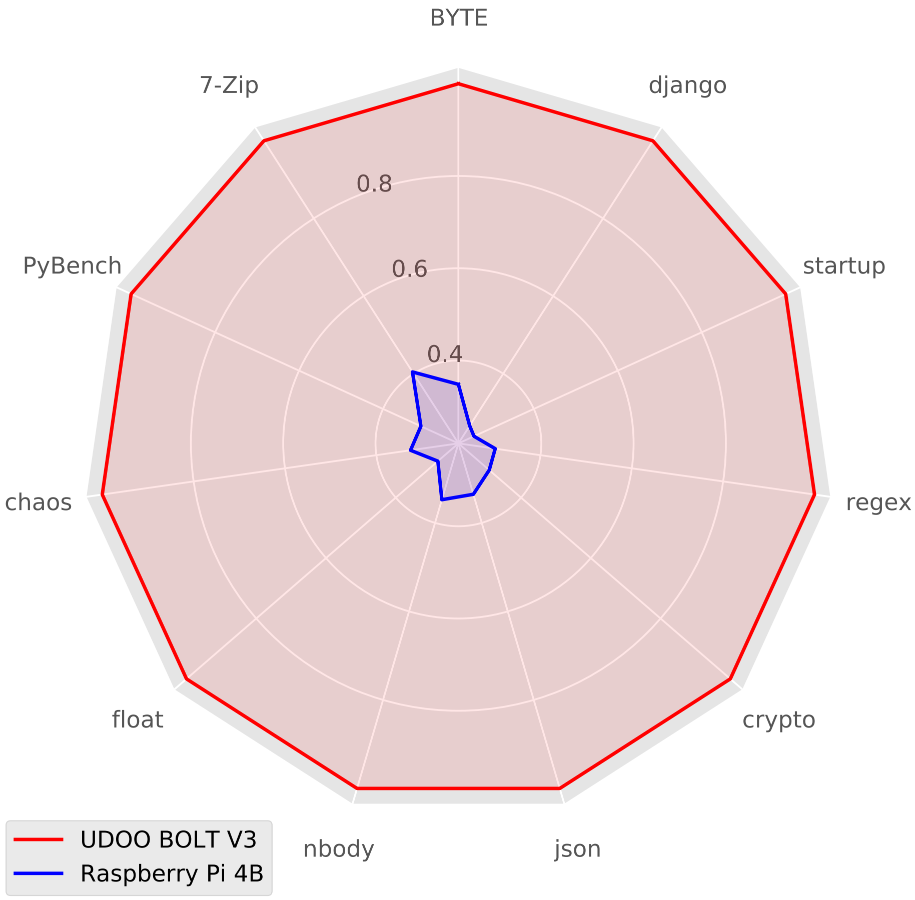


Figure 3.5: Phoronix benchmarks for the UDOO BOLT V3 and Raspberry Pi 4B. The performance values for each test are normalized to UDOO BOLT V3 performance.

From Figure 3.5, we can observe that UDOO BOLT V3 performs much better than Raspberry Pi 4B in every benchmark, as expected. We also notice some disparity in the relative performance throughout the benchmarks, in particular, the 7-Zip benchmark. This is likely due to the fact that 7-Zip benchmark 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). By analyzing the data, we find that UDOO BOLT V3 outperforms Raspberry Pi 4B on average by a factor of 2.21 ± 0.4.

Test 3: MQTT Benchmark

As the *SmartBox* is intended to communicate with the Smart Gateway through MQTT messages, we also evaluate how each system handles the load associated with an MQTT client. For this test, each SBC ran a simple MQTT client, which subscribes to a single topic and publishes a message to another topic at a given transmission rate, with a payload containing the string “hello world”, through an MQTT broker in the local network.

To-do: Repeat tests with higher rates if time allows it.

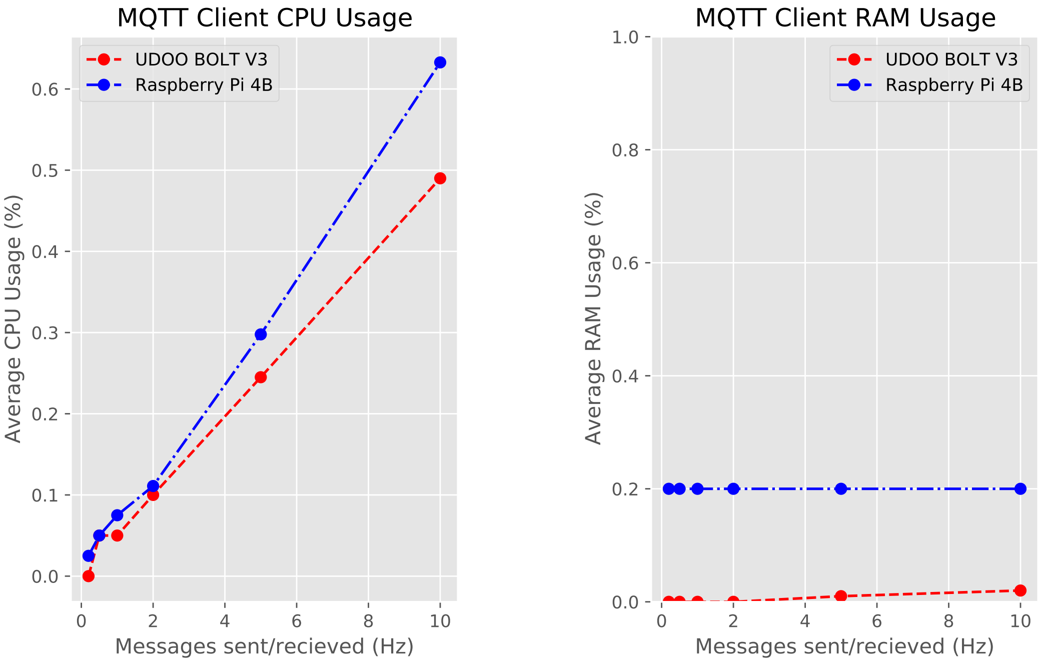


Figure 3.6: MQTT benchmark for Raspberry Pi 4B and UDOO BOLT V3. As the transmission frequency increases, we observe that the Raspberry Pi consumes more resources than the UDOO BOLT V3, nonetheless, these are very negligible performance differences, with an impact of $< 0.5\%$ resource usage.

The MQTT client is a very lightweight process, and as seen in Figure 3.6, it is capable of running on both platforms with trivial performance impact.

3.1.2 Final Decision on SmartBox hardware

Based on the results of our tests we conclude that UDOO BOLT V3 heavily outperforms the Raspberry Pi 4B in CPU benchmarks, but shows a negligible difference in memory usage. Nonetheless, we find that these performance gains do not meaningfully impact the *SmartBox* functionality, for example, in the MQTT communication. We can also extend this to other features, such as the developer GUI or the Python acquisition script, which should have a similar resource load to the MQTT client in the previous MQTT benchmark.

Additionally, the Raspberry Pi 4B comes out of the box with an included WiFi+BLE combo networking card, 8GB of RAM (as we chose the 8GB model), and a much smaller form factor – which is very useful for embedding the *SmartBox* 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 *SmartBoxes*.

Due to all of the aforementioned reasons, we have decided to move forward with the Raspberry Pi 4B for the *SmartBox* development in the scope of the WoW project.

3.2 Communication with the Biostickers

As previously mentioned, the communication between the *Biostickers* and the *SmartBox* 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 finger tip – in order to capture all required biosignals. This means that the BLE acquisition system must be capable of handling both communications simultaneously. Having this in mind, we need to understand if the included BLE adapter of the Raspberry Pi board is sufficient for the task, or if we need to consider a different acquisition hardware.

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 3.7: the Application Layer, the Host Layer and Controller Layer.

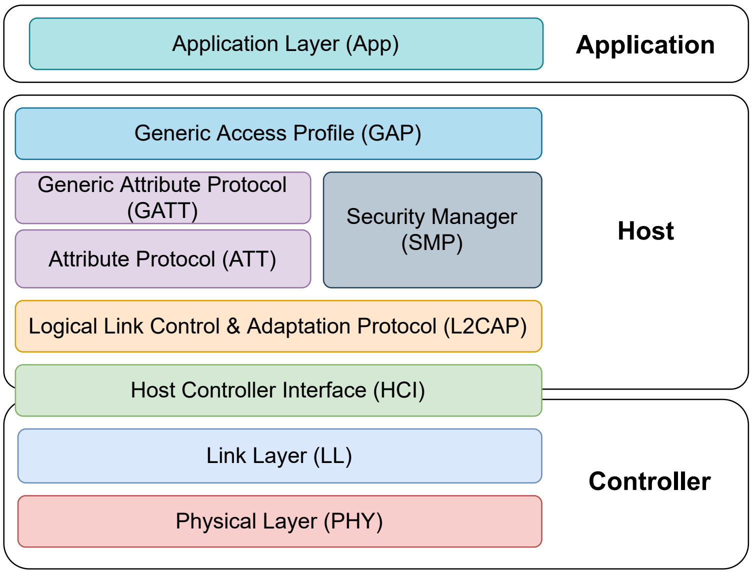

Application Layer (App)
Application
Generic Access Profile (GAP)
Generic Attribute Protocol (GATT)
Attribute Protocol (ATT)
Security Manager (SMP)
Host
Logical Link Control & Adaptation Protocol (L2CAP)
Host Controller Interface (HCI)
Link Layer (LL)
Physical Layer (PHY)
Controller

Figure 3.7: Diagram of the different components of the BLE protocol stack. Adapted from [42, 43]

At the Controller layer, we have the Physical Layer (PHY) and Link Layer (LL) components.

The Physical Layer (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 (ISM) 2.4GHz band. According to the latest revision of the specification [42], BLE supports 3 distinct physical layers: LE 1M, LE 2M and LE Coded. The first is the default PHY, with a data rate of 1 Mbps, which existed in previous versions of Bluetooth, the latter 2 were introduced in Bluetooth 5. LE 2M is an upgrade of the previous PHY, doubling the data rate to 2 Mbps, and LE Coded has a much larger range (up to 4x), at cost of a lower data rate (500 kbps or 125 kbps).

The Link Layer (LL) layer 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.

Afterwards, we have 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 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 Attribute Protocol (ATT).

ATT is the protocol used to expose the application data to other BLE devices through data structured called “Attributes”, which are smallest addressable units of data used by ATT. From the application point of view, data is exchanged using Generic Attribute Protocol (GATT), which can be viewed as a “meta-layed” on top of ATT. Regarding GATT there are 3 main constructs to consider:

- GATT Characteristic – The lowest level concept in GATT transactions is the Characteristic, which encapsulates a single data point, *e.g.* a temperature measurement, a 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 SIG, but can also be a custom profile.

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

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

The data packet format for BLE transmissions is displayed in Figure 3.8, showing how the data is encapsulated for the transmission.

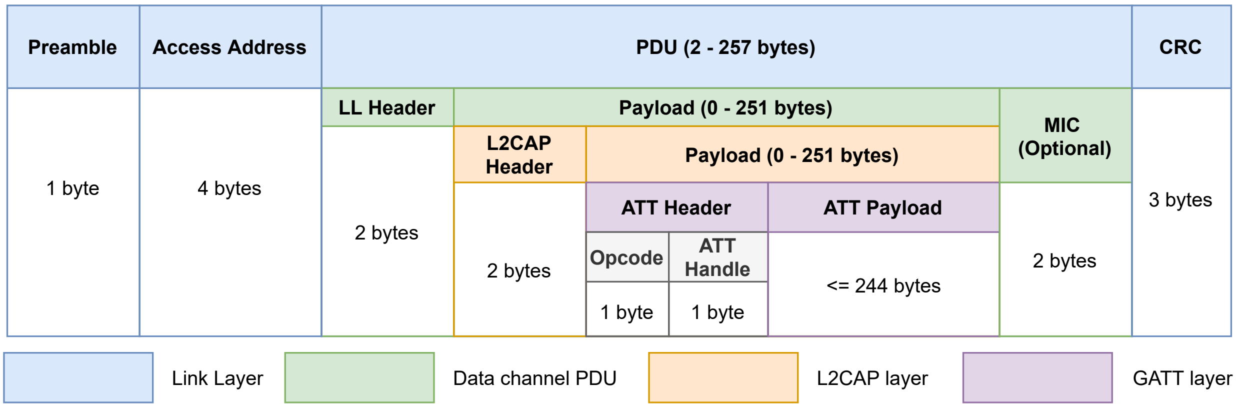


Figure 3.8: BLE data packet format for uncoded PHYs (LE 1M and LE 2M). Adapted from [42, 43]. The Message Integrity Check (MIC) and Cyclic Redundancy Check (CRC), which are not discussed above, are used for validating the integrity of the data packet.

Before we discuss BLE data transmissions, it is important to expose certain terminology which is used in the area:

- 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.

Figure 3.9 shows a single communication event during a BLE communication between two devices, 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. One thing to consider is that, although there are not rules enforcing the Connection Event to have just transmit one data packet, some BLE protocol stack implementation may not support sending more than one packet per Connection Event.

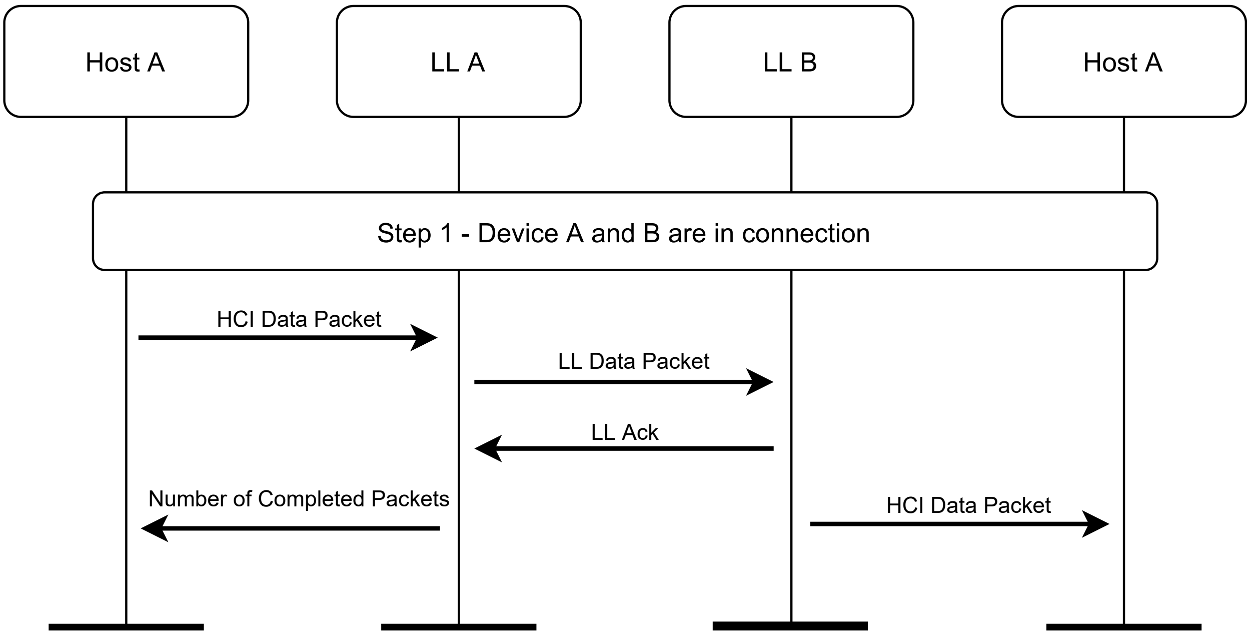


Figure 3.9: Message sequence chart between two BLE devices sending data and replying to the request. Source: [42]

There are multiple parameters which guide how BLE data connections are performed, which are the following:

- Slave Latency: Number of consecutive connection events 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 PDUs 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.

BLE protocol stack on Linux

In the WoW project, the data acquisition is developed using the official Linux implementation of BLE protocol stack[9] – *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, we decided to compare this adapter with a commercially available that met the requirements for the project. The requirements for the adapter are the following:

- The adapter must support, at least, the Bluetooth 5.0 core specification.

- The adapter must natively support Ubuntu 20.04, as well as the *BlueZ* BLE protocol stack.

After researching the available market, we chose the Asus USB-BT500 USB adapter[10] due to its affordability and availability, making it an adequate fit for the project's needs.

[9] *BlueZ*: `http://www.bluez.org/profiles/`

[10] Asus USB-BT500 Bluetooth 5.0 USB Adapter: `https://www.asus.com/Networking-IoT-Servers/Adapters/All-series/USB-BT500/`

In the next section, we perform multiple tests to evaluate the performance of Asus USB-BT500 and the internal BLE adapter in the Raspberry Pi 4B.

3.2.3 Testing BLE Communication

To ensure an BLE adapter is capable of handling the communication on the *SmartBox* side for the WoW project, we created two different tests to evaluate their performance at different distances: a test to evaluate the roundtrip time for a single message (for different sizes), and a test to evaluate the maximum bandwidth achieveable using a packet loss analysis for multiple frequencies.

For these tests, we need to ensure the conditions are very similar to those when acquiring data from the *Biostickers*. To this end, we use the same microcontroller – nRF52832 SoC – which is used in the *Biostickers*, with our own custom firmware for the tests. Therefore, the device which is used for "simulating" the BLE device for the tests is the nRF52-DK developer kit[11]. The firmware for the microcontroller is built using the latest version of MbedOS[12] and can be found here[13].

Test Conditions

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

Connection Interval	7.5ms
Slave Latency	0
Supervisor Timeout	500ms
ATT MTU Length	23 bytes

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

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

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

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

Connection Interval	7.5ms
Slave Latency	0
Supervisor Timeout	500ms
ATT MTU Length	251 bytes

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

Additionally, these tests were conducted indoors, with the devices (the BLE adapter and the microcontroller) in clear view of each other.

Test 1: Roundtrip Time Measurement

In this first test, we measure the roundtrip time of a single ATT data packet on a BLE connection for different payload sizes. For this test, the nRF52-DK board exposed two GATT characteristics: one to be updated by the *SmartBox* (characteristic “A”), and one to which the *SmartBox* subscribed (characteristic “B”). When the *SmartBox* writes on characteristic “A”, the nRF52-DK changed the value of the characteristic “B”, immediately sending a notification to the *SmartBox*. The roundtrip time for our purposes is then the time taken to recieve the notification on characteristic “B” after updating the value of characteristic “A”. Below, are the roundtrip time graphs for different payload sizes obtained for each adapter at different distances. Each test was run 5 times to ensure the results are consistent.

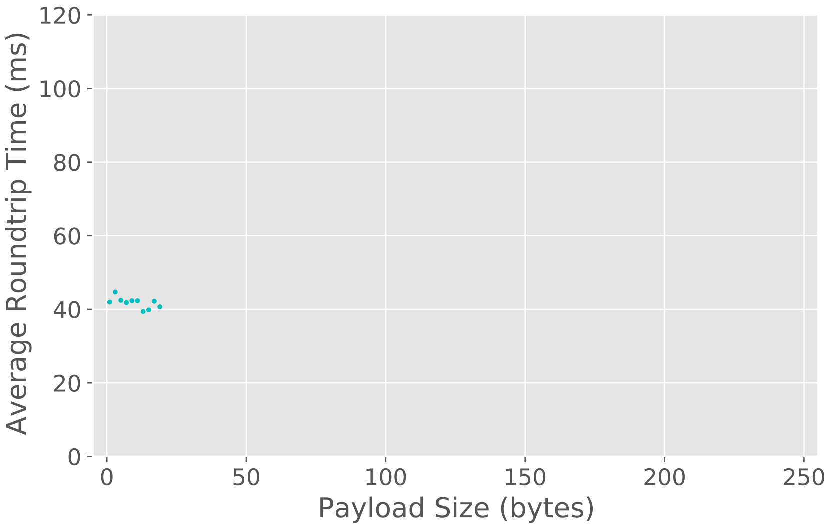


Figure 3.10: Average BLE connection roundtrip time obtained using Raspberry Pi 4B’s internal BLE adapter at a distance of 0m ± 0.2.

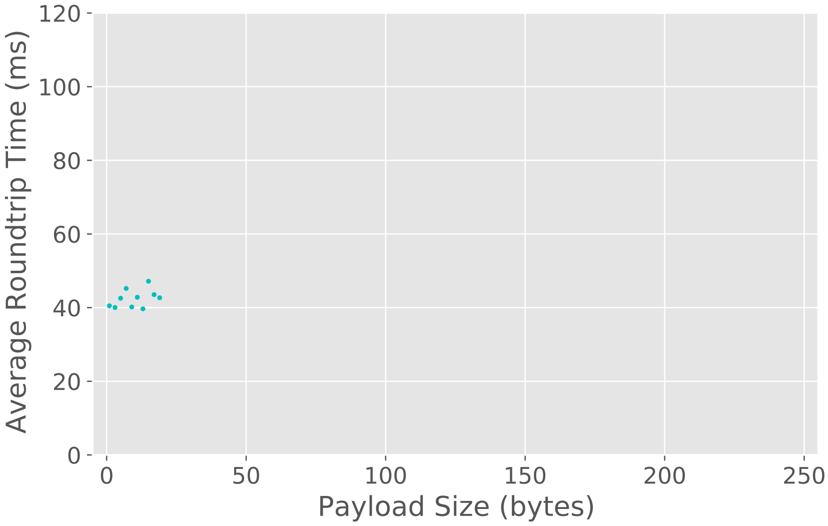


Figure 3.11: Average BLE connection roundtrip time obtained using Raspberry Pi 4B's internal BLE adapter at a distance of 3m ± 0.2.

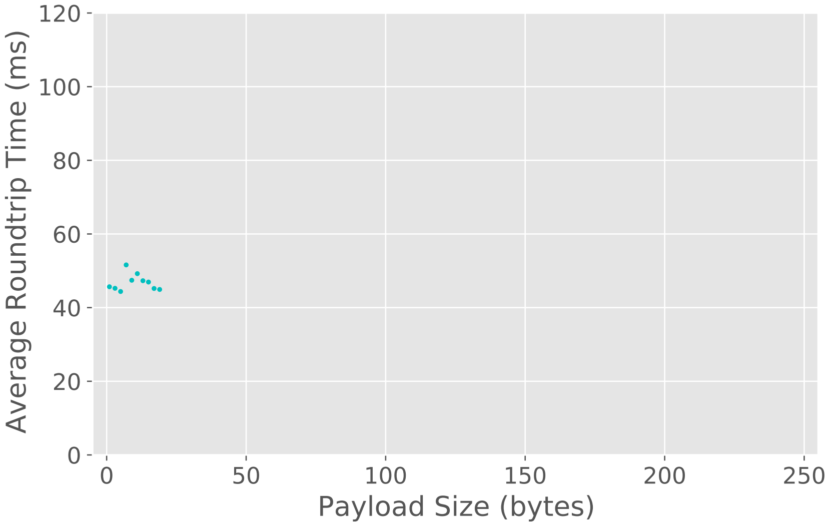


Figure 3.12: Average BLE connection roundtrip time obtained using Raspberry Pi 4B's internal BLE adapter at a distance of 6m ± 0.2.

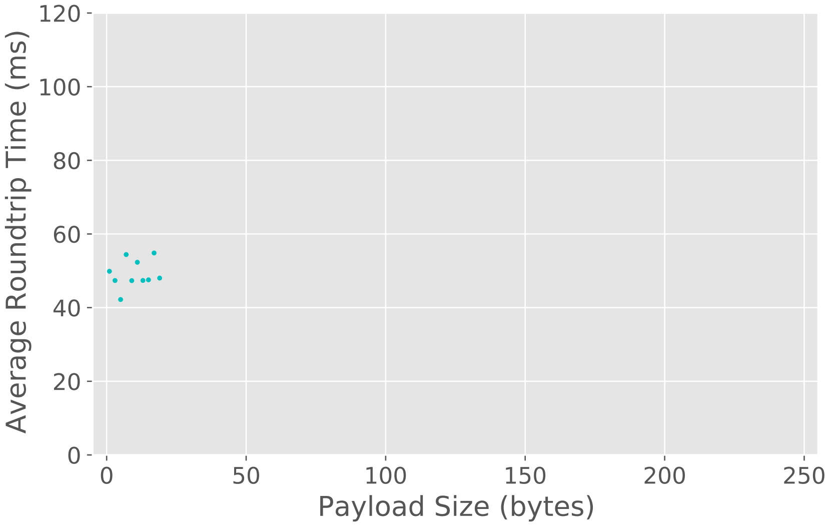


Figure 3.13: Average BLE connection roundtrip time obtained using Raspberry Pi 4B's internal BLE adapter at a distance of 9m ± 0.2.

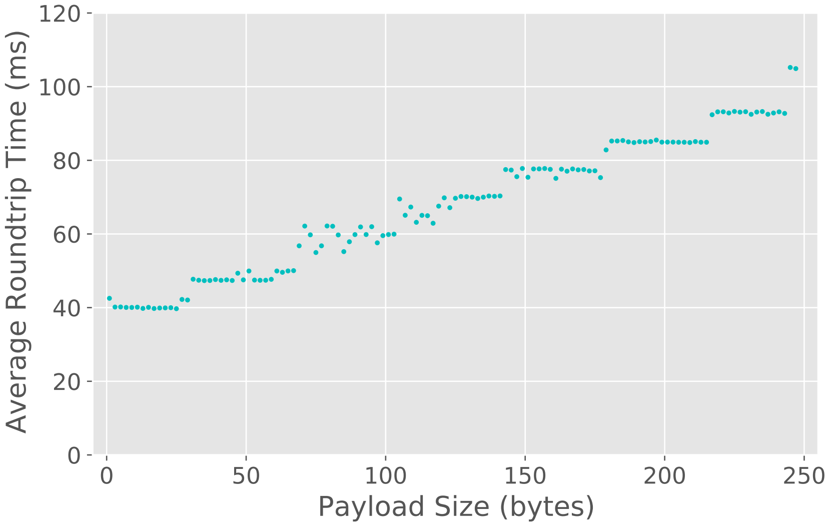


Figure 3.14: Average BLE connection roundtrip time obtained using the Asus USB-BT500 adapter at a distance of 0m ± 0.2.

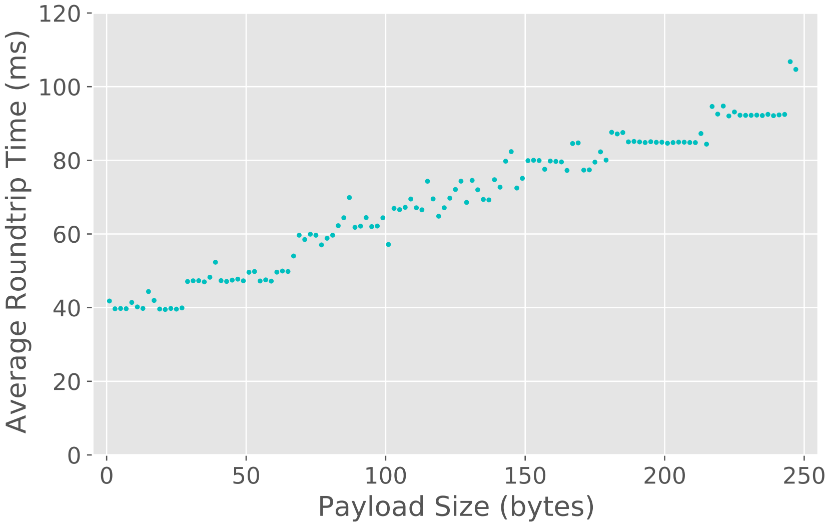


Figure 3.15: Average BLE connection roundtrip time obtained using the Asus USB-BT500 adapter at a distance of 3m ± 0.2.

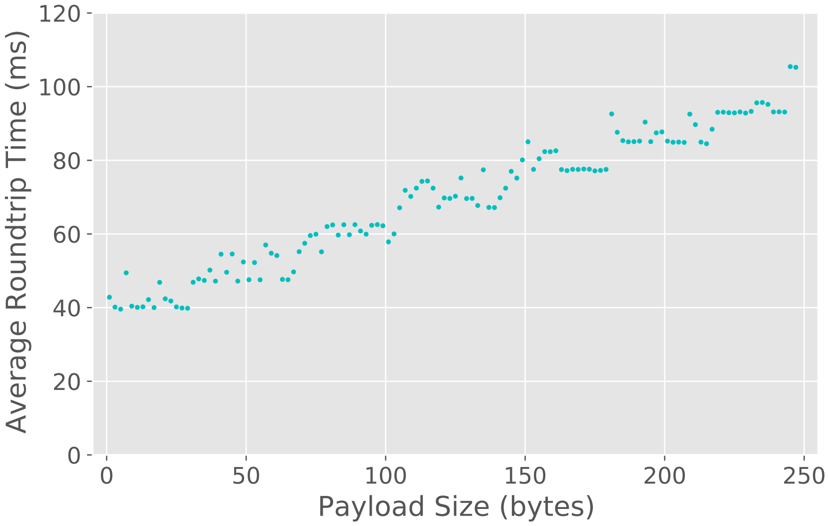


Figure 3.16: Average BLE connection roundtrip time obtained using the Asus USB-BT500 adapter at a distance of 6m ± 0.2.

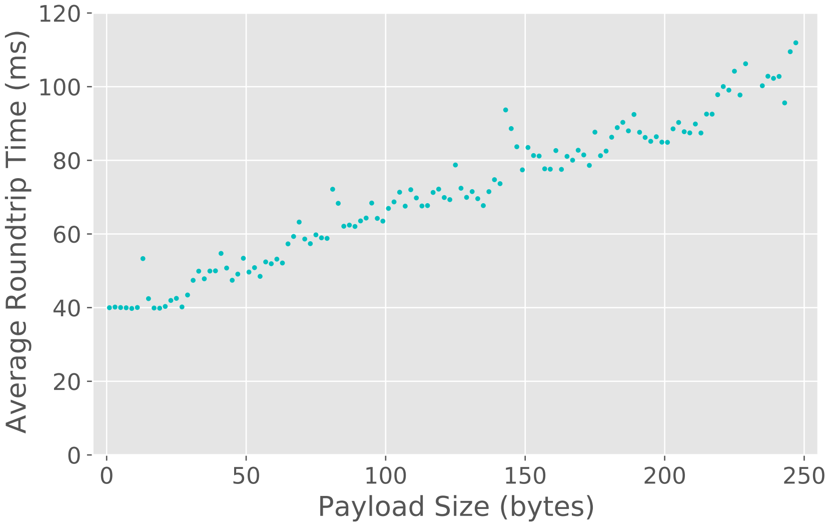


Figure 3.17: Average BLE connection roundtrip time obtained using the Asus USB-BT500 adapter at a distance of 9m ± 0.2.

We find that for both adapters, the minimum roundtrip time is the same: appoximately 40ms. This is an odd value, given that the connection interval was set to 7.5ms, meaning the time it takes to transmit a single message and recieve it on the **SmartBox!** takes nearly x4 connection intervals, nearly double of what should be expected. Nonetheless, it is still a

We can also observe that for both adapters, the roundtrip times measurements become significantively noisy as the distance increases, as expected, although it is particularly noticable for the Raspberry Pi 4B's internal BLE adapter on Figure **??**.

During the development of these tests, we discovered an functionaly which was not documented in the specification – Data Length Extension (DLE). This feature, introduced in Bluetooth 4.2, allows LL data packet payload to increase significantly in size, up to 251 bytes (compared to the 23 bytes when not using this feature). This gives a lead start for the Asus USB-BT500, as it is capable of sending nearly x10 more data in a single transmission, compared to the Raspberry Pi 4B's internal BLE adapter.

Test 2: Bandwidth Measurement

In this second test, we measure the maximum bandwidth for the BLE connection by adjusting the transmission rate, or more accurately, the time between the transmission of each packet.

For this test, the nRF52-DK board exposed 7 GATT characteristics for the *SmartBox* to subscribe to. The nRF52-DK then continuously changes the value of the characteristic, immediately triggering a notification to the *SmartBox* – one for each characteristic. The transmission rate was increased until the connection became to unstable and crashed. Below, are the bandwidth graphs obtained for different transmission rates (which correspond to the frequency with which the characteristics are updated) for each adapter. Each test was run 3 times to ensure the results are consistent.

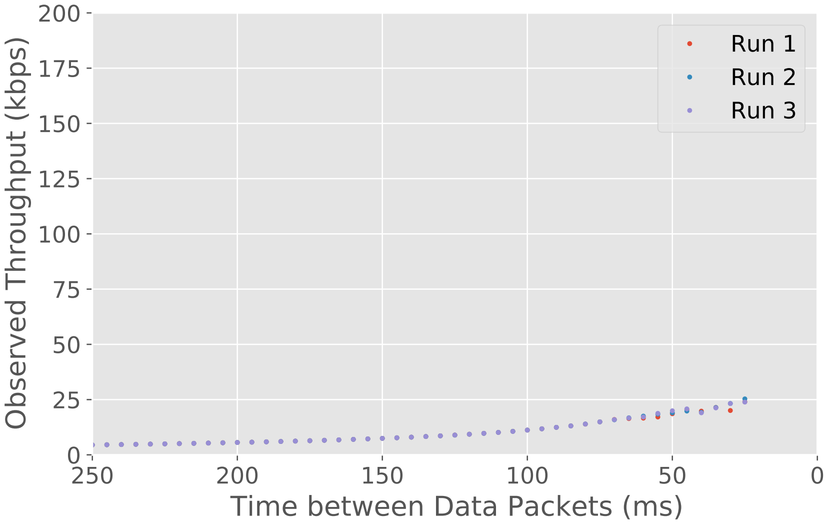


Figure 3.18: BLE connection bandwidth obtained using the Raspberry Pi 4B internal BLE adapter at a distance of 0m ± 0.2.

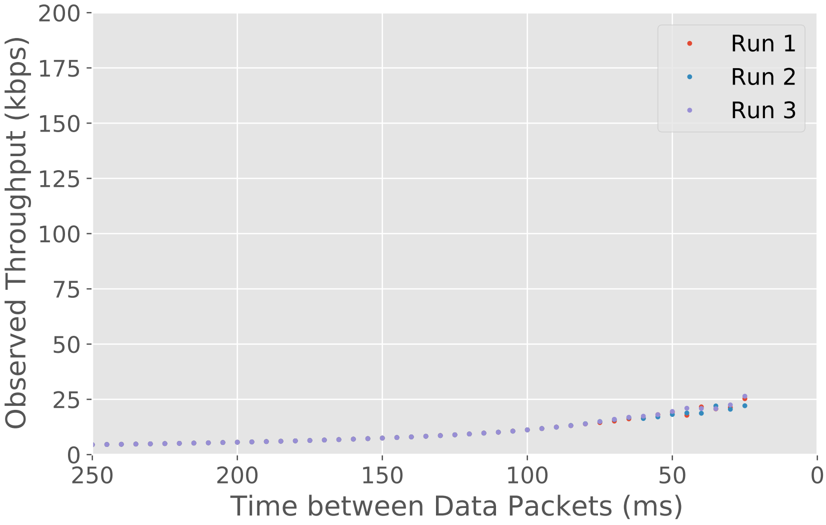


Figure 3.19: BLE connection bandwidth obtained using the Raspberry Pi 4B internal BLE adapter at a distance of 3m ± 0.2.

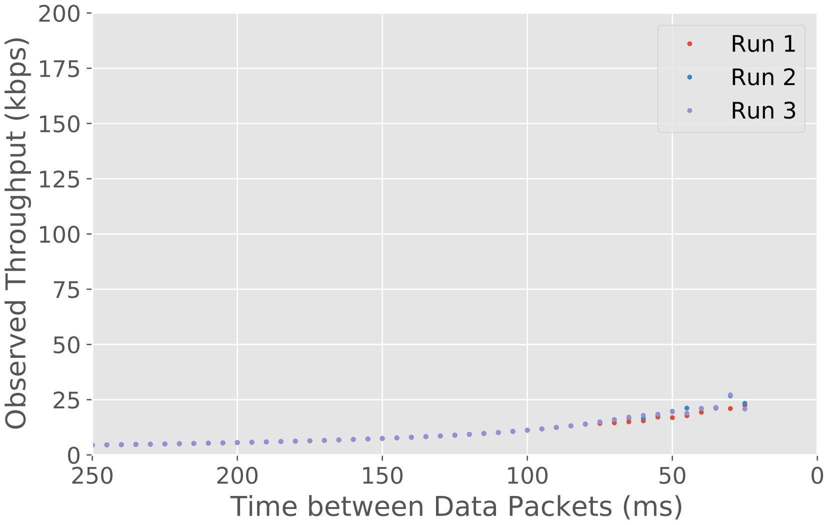


Figure 3.20: BLE connection bandwidth obtained using the Raspberry Pi 4B internal BLE adapter at a distance of 6m ± 0.2.

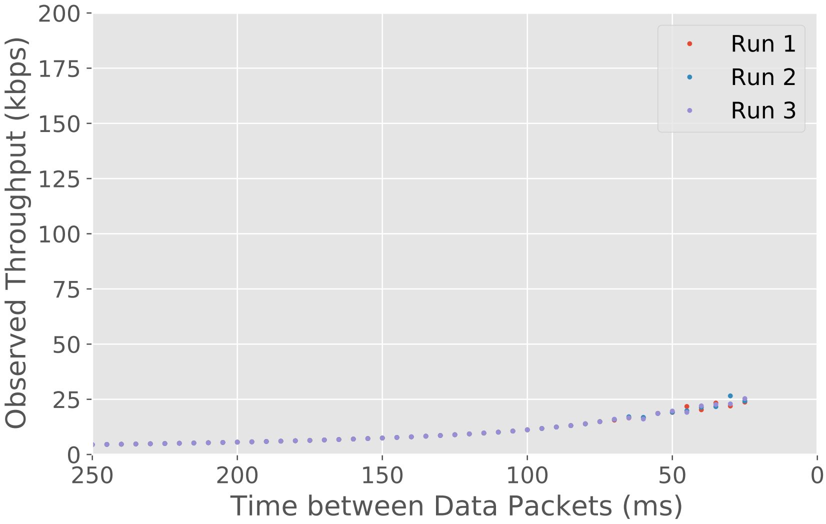


Figure 3.21: BLE connection bandwidth obtained using the Raspberry Pi 4B internal BLE adapter at a distance of 9m ± 0.2.

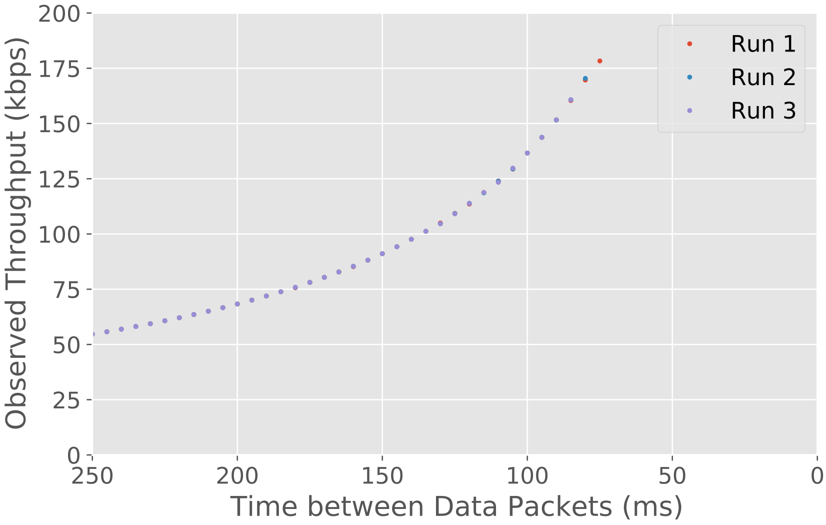


Figure 3.22: BLE connection bandwidth obtained using the Asus USB-BT500 adapter at a distance of 0m ± 0.2.

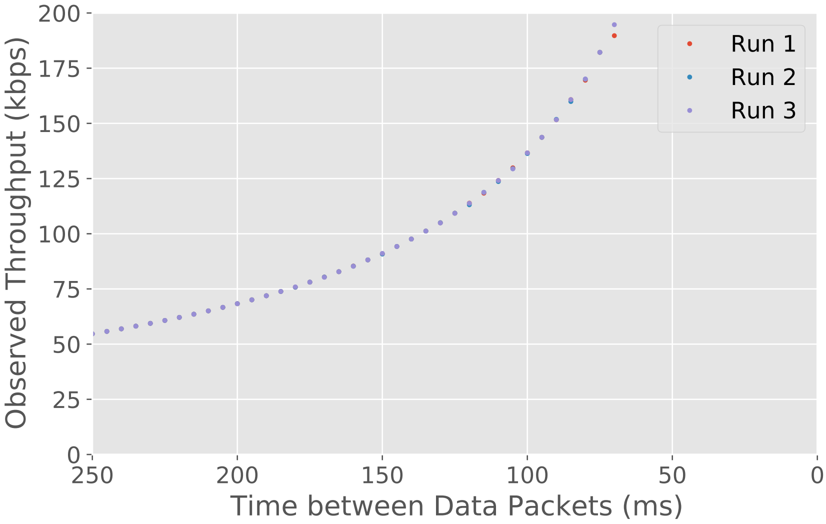


Figure 3.23: BLE connection bandwidth obtained using the Asus USB-BT500 adapter at a distance of 3m ± 0.2.

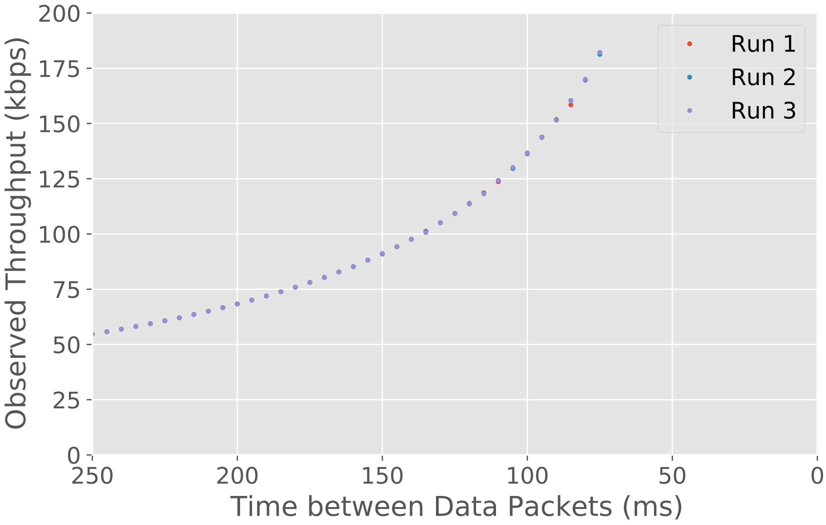


Figure 3.24: BLE connection bandwidth obtained using the Asus USB-BT500 adapter at a distance of 6m ± 0.2.

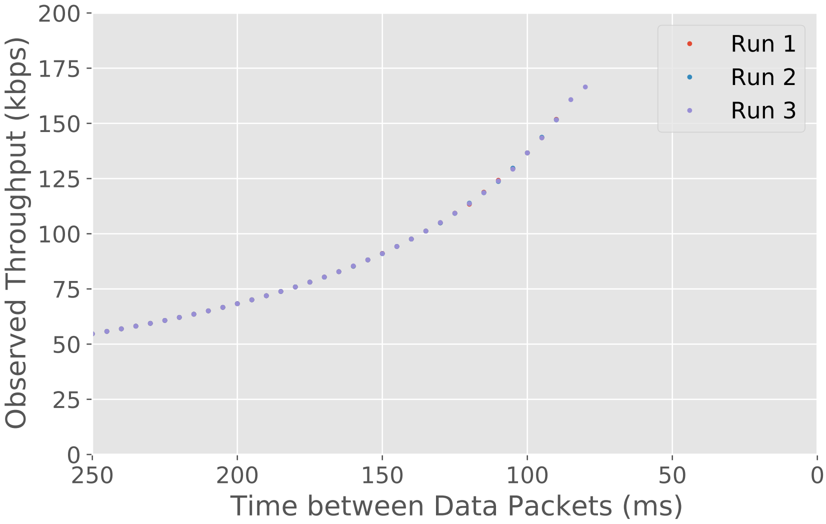


Figure 3.25: BLE connection bandwidth obtained using the Asus USB-BT500 adapter at a distance of 9m ± 0.2.

3.2.4 Decision on BLE adapter

From the previous tests, we see that the internal BLE adapter lacks the support for DLE feature, which reduces greatly the communication throughput.

3.3 Summary

In this chapter, we ...

4 Smart Gateway Development

In the proposed architecture, the Gateway is the central module of the system, connecting the SmartBoxes to the HIS. It's also responsible for the management of devices and their associations (SmartBox - Biosticker), managing, maintaining and storing the data that is generated by these, as well as handling any communication from the HIS. The Gateway maintains a list of all the SmartBoxes that are managed by the system, as well as every Biosticker and every sensor in the biosticker (which are used to indicate which "sensor" measured the respective biosignal to the HIS).

4.1 Services Overview

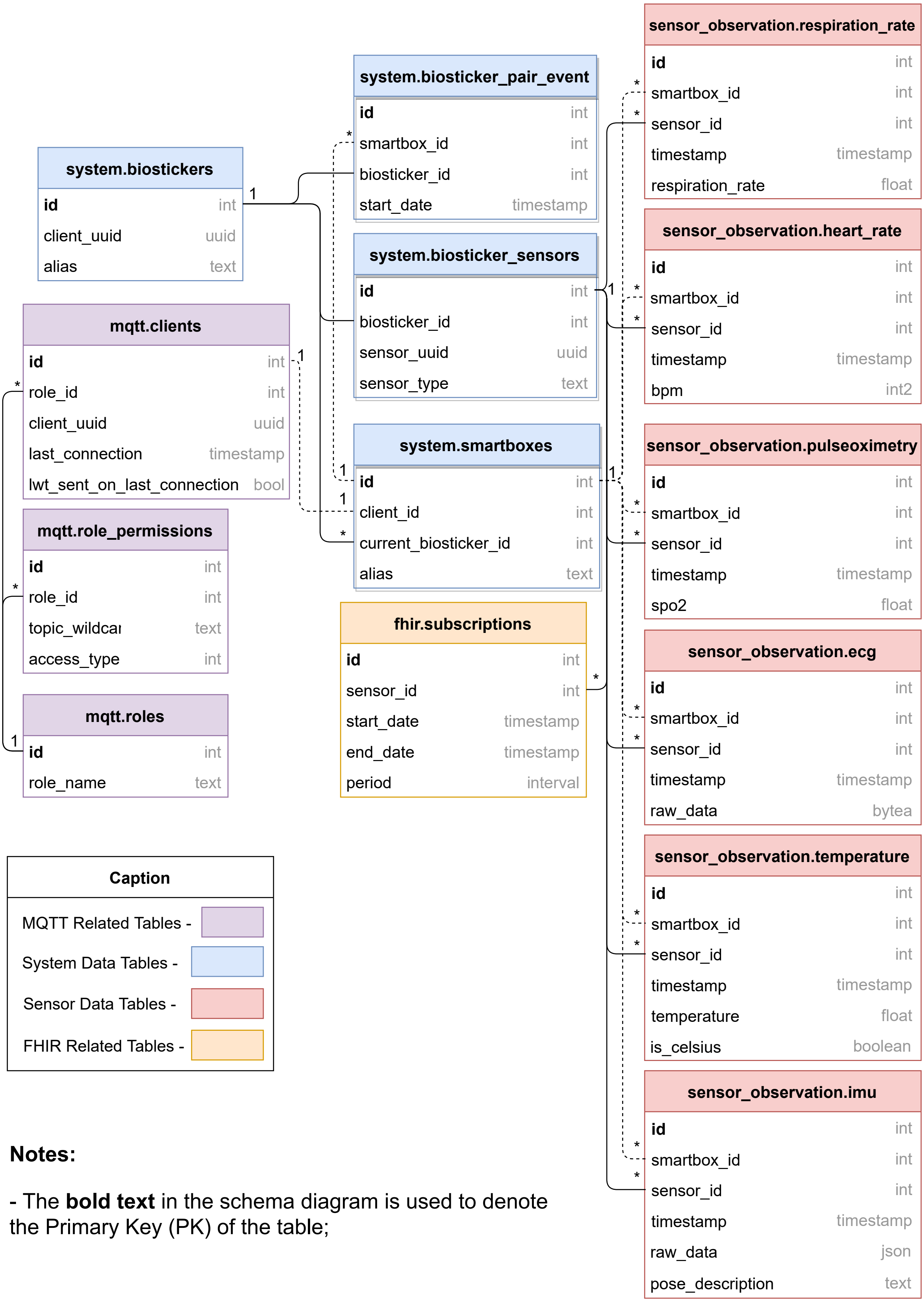
...

4.2 Data Store

...

4.2.1 Database Schema

...



system.biostickers	
id	int
client_uuid	uuid
alias	text

mqtt.clients	
id	int
role_id	int
client_uuid	uuid
last_connection	timestamp
lwt_sent_on_last_connection	bool

mqtt.role_permissions	
id	int
role_id	int
topic_wildcar	text
access_type	int

mqtt.roles	
id	int
role_name	text

system.biosticker_pair_event	
id	int
smartbox_id	int
biosticker_id	int
start_date	timestamp

system.biosticker_sensors	
id	int
biosticker_id	int
sensor_uuid	uuid
sensor_type	text

system.smartboxes	
id	int
client_id	int
current_biosticker_id	int
alias	text

fhir.subscriptions	
id	int
sensor_id	int
start_date	timestamp
end_date	timestamp
period	interval

sensor_observation.respiration_rate	
id	int
smartbox_id	int
sensor_id	int
timestamp	timestamp
respiration_rate	float

sensor_observation.heart_rate	
id	int
smartbox_id	int
sensor_id	int
timestamp	timestamp
bpm	int2

sensor_observation.pulseoximetry	
id	int
smartbox_id	int
sensor_id	int
timestamp	timestamp
spo2	float

sensor_observation.ecg	
id	int
smartbox_id	int
sensor_id	int
timestamp	timestamp
raw_data	bytea

sensor_observation.temperature	
id	int
smartbox_id	int
sensor_id	int
timestamp	timestamp
temperature	float
is_celsius	boolean

sensor_observation.imu	
id	int
smartbox_id	int
sensor_id	int
timestamp	timestamp
raw_data	json
pose_description	text

Caption	
MQTT Related Tables -	🟪
System Data Tables -	🟦
Sensor Data Tables -	🟥
FHIR Related Tables -	🟧

Notes:

- The **bold text** in the schema diagram is used to denote the Primary Key (PK) of the table;

Figure 4.1: test

MQTT Client Information

. . .

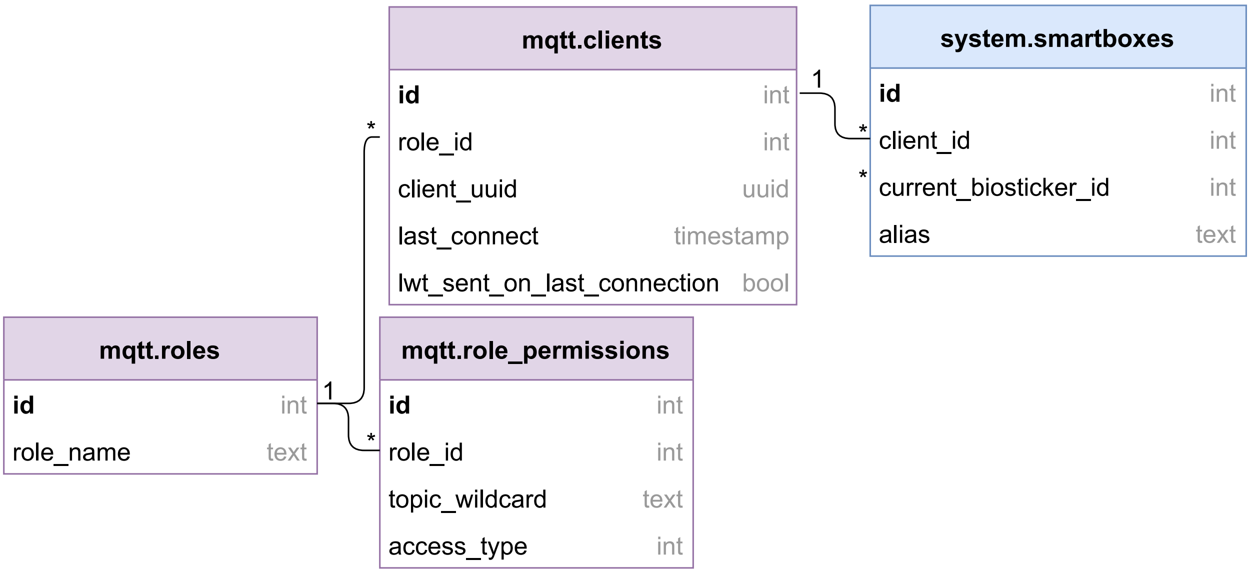


Figure 4.2: test

Sensor Data

. . .

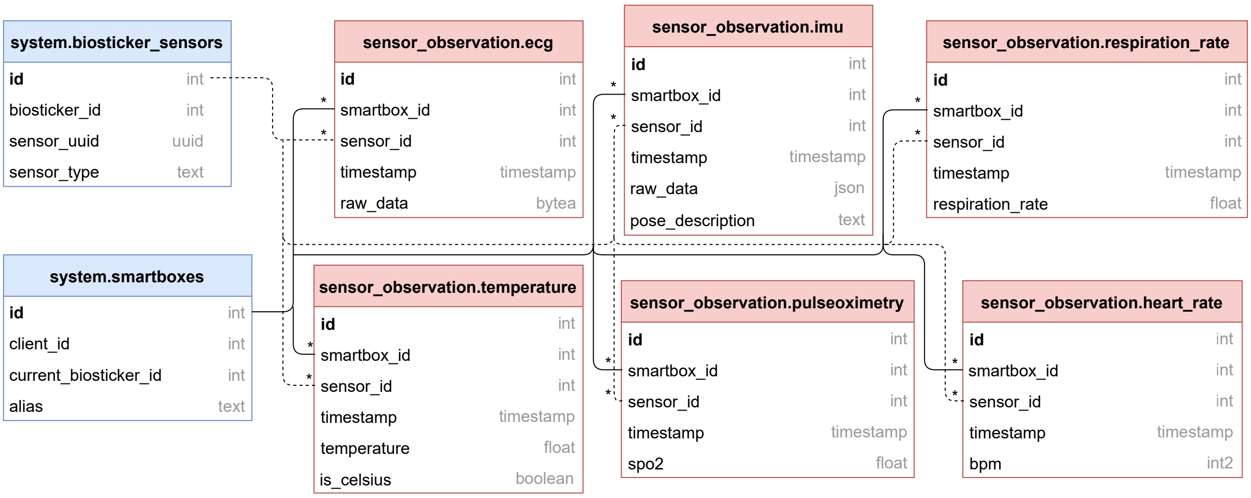


Figure 4.3: test

FHIR Related Information

. . .

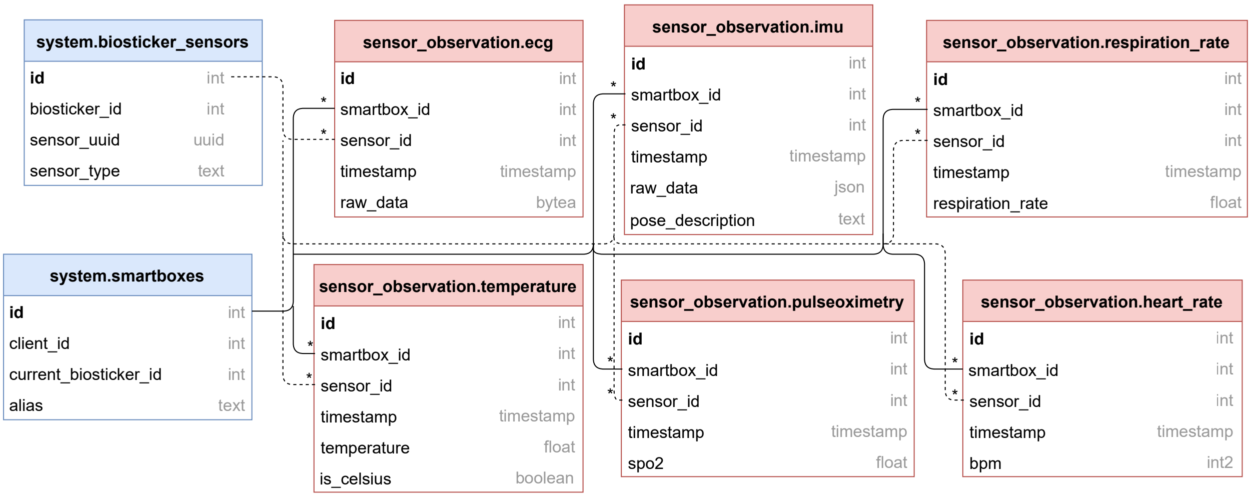


Figure 4.4: test

4.3 Connection to the SmartBoxes

4.4 Integration with GlobalCare

4.5 Summary

In this chapter, we ...

5 Results and Discussion

...

5.1 Hospital Pilot

...

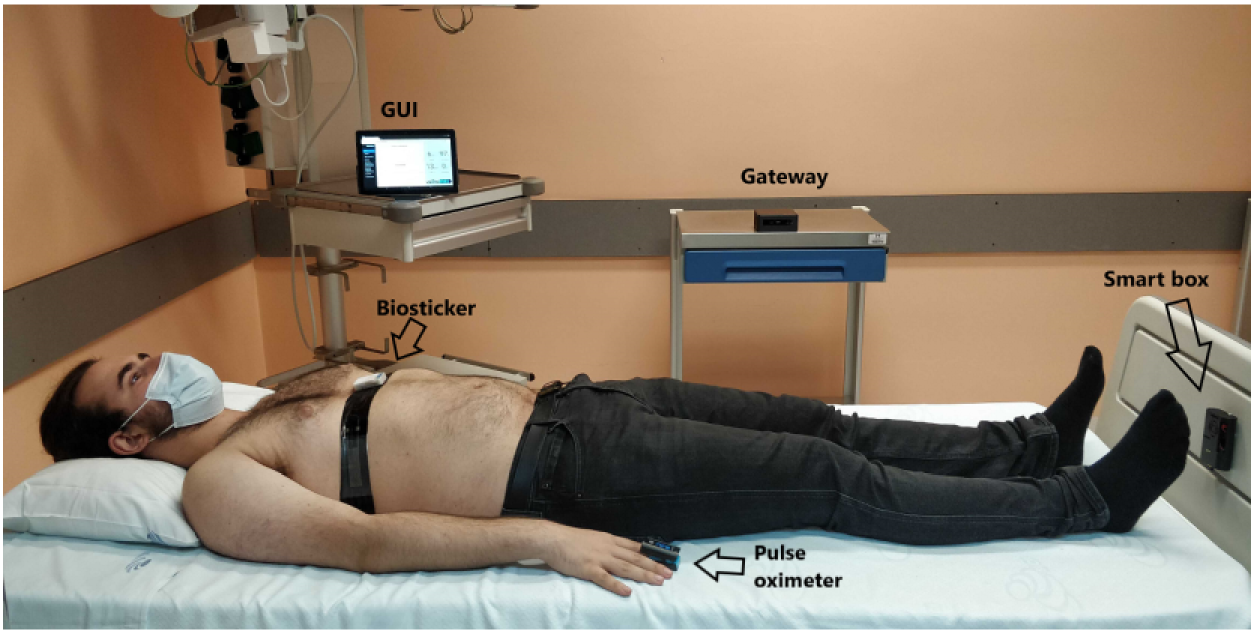


Figure 5.1: Conceptual illustration of the system components within a medical facility.

...

5.2 Results

...

5.3 Summary

In this chapter, we ...

6 Conclusion

...

6.1 Future Work

...

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. [Online]. Available: https://doi.org/10.1007/s41999-019-00219-8

[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. [Online]. Available: http://dx.doi.org/10.1016/j.adhoc.2012.04.006https://linkinghub.elsevier.com/retrieve/pii/S1570870512000777

[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. 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] 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, no. xxxx, p. 102845, 2020. [Online]. Available: https://doi.org/10.1016/j.iccn.2020.102845

[9] 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.

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

[11] 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

[12] 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.

[13] 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/

[14] 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.

[15] 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. [Online]. Available: http://ieeexplore.ieee.org/document/6722995/

[16] 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. [Online]. Available: http://ieeexplore.ieee.org/document/7070665/

[17] 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. [Online]. Available: http://dx.doi.org/10.1016/j.future.2016.12.023https://linkinghub.elsevier.com/retrieve/pii/S0167739X16308007

[18] 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.

[19] 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

[20] 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. [Online]. Available: https://ieeexplore.ieee.org/document/8767280/

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

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

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

[30] 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.

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

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

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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

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

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

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

[41] 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.

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

[43] 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.