FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Remote Biometrical Monitoring System via IoT

Pedro de Castro Albergaria

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Prof. Dr. Luís Miguel Pinho de Almeida

Second Supervisor: Prof. Dr. Pedro Miguel Salgueiro dos Santos

July 30th, 2020

Resumo

As instituições de cuidados de saúde tentam providenciar os melhores serviços no que refere à fiabilidade, segurança e conforto dos seus pacientes. Nos últimos anos, a tecnologia *Internet of Things* (IoT) tem sido adotada e desenvolvida para melhorar estes serviços.

Os sistemas IoT têm registado um rápido crescimento devido à sua aplicabilidade em diversos domínios, desde de cidades inteligentes a cuidados de saúde. Nestes sistemas, os diposi-tivos comunicam entre si, ou com a infraestrutura, através de comunicações *machine-to-machine* (M2M). Devido ao facto de que muitos destes dispositivos possuem poucos recursos computacionais, vários protocolos M2M foram desenvolvidos como o *Constrained Application Protocol* (CoAP) e o *Messaging Queue Telemetry Transport* (MQTT). Existem diversos desafios no desenvolvimento de aplicações M2M e IoT, entre eles a interoperabilidade e *standardisation*. Por isso, vários *standards* M2M foram desenvolvidos para resolver estes problemas, sendo o oneM2M um deles. Atualmente, há vários dispositivos presentes no mercado com uma antena WiFi embebida, o que permite a integração destes dispositivos num sistema IoT sem a necessidade de um dispositvo com a funcionalidade de *gateway* (GW) para a ligação à Internet. O módulo ESP32 da Espressif é um dos módulos presentes no mercado que suporta a tecnologia WiFi bem como possui modos de mecanismo de poupança de energia relacionados com esta.

O trabalho proposto nesta dissertação consiste num sistema IoT ponta-a-ponta baseado num dispositivo de baixo custo e baixo consumo que suporta a tecnologia WiFi capaz de monitorizar continuamente os parâmetros fisiológicos do usário e apresentá-los, por exemplo, a profissionais de saúde. O sistema pode ser aplicado em diversos casos de uso como alas de emergência e competições desportivas.

O sistema possui duas componentes principais: um dispositivo wearable baseado num módulo WiFi de baixo custo e baixo consumo para o usuário, e uma aplicação de monitorização com uma interface gráfica direcionada aos profissionais de saúde. O primeiro componente é constituído por sensor fotopletismográfico MAX30102 para medir o batimento cardíaco e saturação de oxigénio no sangue, um ESP32 para processar e enviar os dados do usuário para a aplicação de monitorização e, por último, uma bateria de Lítio-polímero (LiPo) para fornecer energia aos outros dois componentes. A aplicação de monitorização é composta por uma base de dados desenhada para eventos temporais (InfluxDB) com a funcionalidade de armazenar os dados do sistema, uma ferramenta de visualização de dados (Chronograf) e uma aplicação de monitorização com uma interface gráfica que serve como painel de controlo.

Adicionalmente, o sistema assenta no *standard* oneM2M e segue o modelo de comunicação publicador-subscritor devido à eficiência deste na monitorização remota. Foram implementados e comparados três protocolos M2M: CoAP, HTTP e MQTT. De modo a avaliar a *performance* do sistema, foram conduzidas experiências de latência ponta-a-ponta (E2E), com transmissão de dados a diferentes frequências, diferentes modos de poupança de energia do ESP32 e diferentes protocolos de comunicação para dois canais de WiFi com qualidade de sinal diferente. Os resultados demonstram que a latência E2E para o canal de WiFi com má qualidade de sinal aumenta

30.2% e 38.2% para mensagens publicadas com um período de 1 e 10 segundos com tamanhos de mensagem de 85B e 850B, respetivamente, quando comparado à latência E2E experienciada por um canal de WiFi com boa qualidade de sinal. É possível concluir que os protocolos baseados em TCP, como o HTTP e MQTT, experenciam um aumento da latência E2E quando comparados ao CoAP, que é um protocolo baseado em UDP. Adicionalmente, para as medidas com tamanho de mensagem de 850B, há uma aumento da latência E2E de 26.3% quando comparada às medidas de 85B entre os dois canais de WiFi. Por fim, relativamente ao *Packet Delivery Ratio* (PDR), concluímos que este é de 100% para o canal WiFi de boa qualidade de sinal, mas para o canal de WiFi de má qualidade, o valor do PDR diminui quando ESP32 publica as mensagens em modos de poupança de energia mais eficientes.

Nesta dissertação, foi desenvolvido um sistema de monitorização focado no baixo custo e eficiência energética. No entanto, não compromente a fiabilidade e robustez de sistemas de monitorização tradicionais.

Palavras-chave: Monitorização Biométrica, CoAP, HTTP, IoT, M2M, MQTT.

Abstract

Healthcare institutions always strive to provide the best services concerning the reliability, safety and comfort of the patients. To do so, Internet of Things (IoT) technologies have been embraced and developed in recent years to improve these services.

IoT systems are experiencing rapid growth due to their applicability in several domains, from smart cities to healthcare, among many. In these systems, devices communicate with each other, or with infrastructure, resorting to machine-to-machine (M2M) communications. Since many of these devices are resource-constrained and have limited computing capabilities, lightweight M2M protocols were developed such as Constrained Application Protocol (CoAP) and Messaging Queue Telemetry Transport (MQTT) and accompanying frameworks to support the operation of the protocols, and promote integration with the target systems. There are challenges when developing M2M and IoT applications: interoperability, scalability, standardisation, among others. Several M2M standards were designed to address these issues, with oneM2M being one of them. Nowadays, there are multiple devices available that have an embedded WiFi interface, thus eschewing the need for a GW in order to be integrated in a IoT architecture to access the Internet since WiFi is one of the most common technologies at Internet boundary. This is a key feature because it increases the system's pervasiveness while decreasing the overall cost of the system. Additionally, these devices, such as the Espressif ESP32 module, offer power management modes that allow exploiting the power management features by the IEEE 802.11 standard.

The work proposed in this dissertation is an end-to-end IoT-inspired system based on small form-factor, ultra-low power and WiFi enabled embedded devices capable of continuously monitoring a user's vital signs and displaying them, e.g., to medical personnel. Such system can be applied to a wide range of application scenarios from emergency wards and home environment to sports training and competition.

The system has two major components, a low-cost low-power WiFi-enabled wearable device for the user and a monitoring application for the medical personnel. The wearable is composed by a MAX30102 PhotoPletysmoGraphy (PPG) sensor to measure the heart rate and oxygen saturation levels, an ESP32 with a built-in WiFi antenna to process and send the sensor data to the monitoring system and, finally, a Lithium Polymer (LiPo) battery to power the previous two components. The monitoring application is composed of a time-series database (InfluxDB) to store all the data, a graphics visualisation software (Chronograf) for displaying user's vital signs and a monitoring application with a Graphical User Interface (GUI) serving as a control panel.

Additionally, the system relies on the oneM2M standard for the interoperability concerning the architecture and follows a publish-subscribe communication model due to its efficiency in sensing and remote monitoring. Also, we implement and compare three M2M protocols, namely CoAP, HTTP and MQTT. To evaluate the system's performance, we conducted end-to-end (E2E) latency experiments, with data transmissions at different frequencies, using the different power modes offered by the ESP32 module and different communication protocols for two WiFi channel quality experiments regarding signal strength. The results show that the E2E latency for a *bad*

WiFi channel quality increases 30.2% and 38.2% for messages published with 1 second and 10 seconds periods with 85B and 850B of payload, respectively, when compared to a *good* WiFi channel quality. We also conclude that scenarios with TCP-based protocols, such as the HTTP and MQTT, experience more E2E delay than UDP-based protocols, i.e. CoAP. Moreover, for the 850B of payload measures, there is an increase of 26.3% when comparing to 85B of payload for the two channels experiments. Moreover, we conclude that the PDR is 100% in the *good* channel experiment while in the *bad* experiment there is a decrease of the PDR when the ESP32 is using more energy-efficient power modes.

Furthermore, in this dissertation we developed a low-cost and energy-efficient monitoring system while not compromising the reliability and robustness of traditional machines and systems.

Keywords: Biometrical Monitoring, CoAP, HTTP, IoT, M2M, MQTT.

Acknowledgements

First, I would like show my gratitude towards my supervisors, Professor Luís Almeida and Professor Pedro Santos, for continuously giving me support throughout the all dissertation process.

I would also like to thank my family, namely my parents, sisters and grandmother for all the love and support.

To all my friends, a sincere thank you for always being there for me. Specially, I want thank to my colleague and dear friend Daniel Silva, for his direct involvement at the early stages of the dissertation.

Pedro de Castro Albergaria

"Life can only be understood backwards;
but it must be lived forwards"

Soren Kierkegaard

Contents

List of Figures

List of Tables

Abbreviations and Symbols

ADC	Analog-to-Digital Converter
ADN	Application Dedicated Node
AE	Application Entity
ALP	Application Layer Protocol
AP	Access Point
API	Application Programming Interface
AQMP	Advanced Queuing Message Protocol
BPM	Beats Per Minute
CoAP	Constrained Application Protocol
CVD	Cardiovascular disease
CSE	Common Service Entity
DDS	Data Distribution Service
DTIM	Delivery Traffic Indication Message
E2E	End-to-end
GUI	Graphical User Interface
HR	Heart-rate
HTTP	Hypertext Transfer Protocol
I^2C	Inter-Integrated Circuit protocol
ID	Identifier
IETF	Internet Engineering Task Force
IoT	Internet Of Things
LiPo	Lithium-ion Polymer
Mbps	Mega bits per seconds
MIPS	Million Instructions Per Second
MQTT	Message Queuing Telemetry Transport
N/A	Not Admitted
NSE	Network Services Entity
OCF	Open Connectivity Foundation
OS	Operating System
PDR	Packet Delivery Ratio
PPG	PhotoPletysmoGraphy
QSPI	Quad Serial Peripheral Interface
QoS	Quality of Service
TCP	Transport Control Protocol
UDP	User Datagram Protocol
ULP	Ultra-Low-Power
URI	Universal Resource Identifier
RSSI	Received Signal Strenght Indication

RTC	Real-Time Clock
RTOS	Real-Time Operating System
SCADA	Supervisory Control And Data Acquisition
SCL	Serial Clock Line
SDA	Serial Data Line
SDK	Software Development Kit
SoC	State of Charge
S_pO_2	Blood's Oxigen Level
STA	Station
SHF	Super High Frequency
WHO	World Health Organization
XMPP	Extensible Messaging and Presence Protocol

Chapter 1

Introduction

1.1 Context and Motivation

According to the World Health Organization (WHO), the global life expectancy increased all over the world in the past two decades, from 67.5 years in 2000 to 72 years in 2016 [1]. This increase stems from better healthcare services offered to the mass population. On the other hand, cardiovascular diseases (CVDs), such as strokes and heart attacks, are the main cause of death today, taking an estimate of 17.9 million lives per year (which represent 31% of the worldwide death toll) [2]. The individuals at risk of a CVD need continuous care and monitoring of their physiological parameters.

In recent years, new technology solutions have been developed for a more convenient way of monitoring patient's vital signs in hospital environment and at home, while offering the same quality of service. Namely, the Internet of Things (IoT) technological paradigm has evolved rapidly to accommodate this use case. Throughout the years, numerous IoT architectures have been proposed to this domain, with small and comfortable wearable devices to acquire vital signs from the patient and display them to the medical personnel. Moreover, these implementations allow the patient to be monitored from home through the Internet, without having the need to go to the hospital or other healthcare facility. However, most monitoring applications in the market with wearables devices capable of continuous monitoring of physiological rely on short-range communication technologies, typically Bluetooth. This entails the need of an extra device an extra device (for example, a smartphone) serving as a gateway (GW) for sustained Internet connection through another technology, e.g., WiFi or cellular. Although the smartphones present enhanced connectivity, the additional GW implementation increases the power consumption of the device which decreases its autonomy and poses an additional load burden to the user.

The main motivation to carry out of this dissertation is the implementation and assessment of an open IoT architecture for continuous monitoring applied to the real-time tracking of the physiological parameters of the users, such as the heart rate, and displaying the data to medical personnel. We follow the approach proposed in [3] relying on the ESP32 module that includes WiFi connectivity and runs a full TCP/IP, allowing to bypass the referred GW.

Figure 1.1 illustrates an application scenario in an emergency ward.

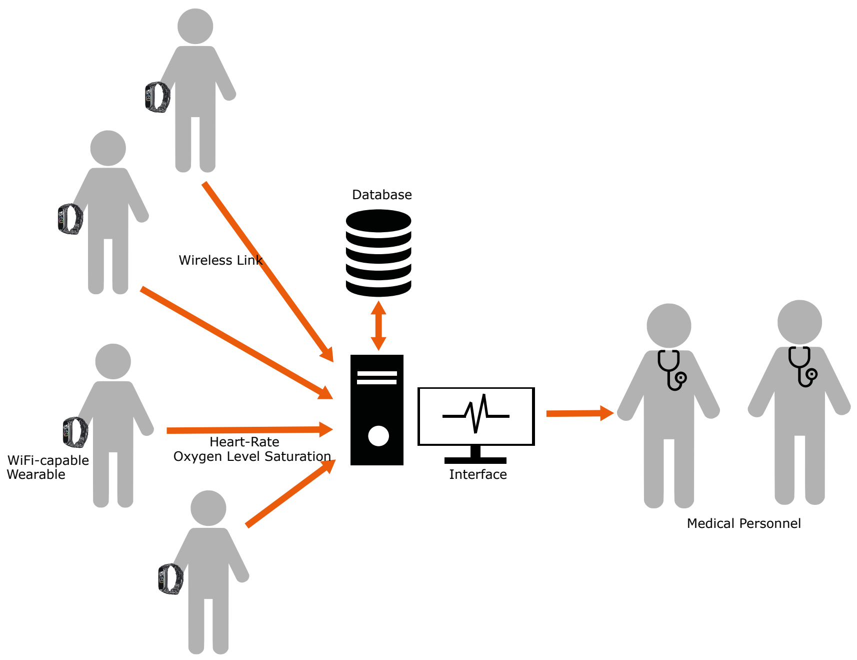


Figure 1.1: Application scenario in an emergency ward. Physiological parameters (heart rate and oxygen saturation levels) are collected by the wearables from patients and forwarded to he monitoring system, where can be accessed by the medical personnel.

1.2 Objectives

The main objective of the work covered by this dissertation is to implement a real-time open IoT architecture for continuous monitoring of physiological parameters of the users with low-cost components and test the system in a sports competition game as well as home monitoring. In particular, develop a WiFi-enabled wearable sensor device based on the Espressif ESP32 module. The wearable device also contains a MAX30102 PhotoPlethymosgraphy sensor for acquiring heart-rate (HR) and blood's oxygen level (S_pO_2), and a LiPo to power the former components. Additionally, the system must present the patient data to the medical personnel, thus developing a monitoring application with real-time data is required. Lastly, we conducted end-to-end (E2E) latency experiments to compare the ESP32's performance in different power modes by publishing different message payloads for two WiFi channel qualities regarding it's Received Signal Strength Indication (RSSI), *good* signal strength channel and *bad* signal strength channel. Moreover, we also assessed the Packet Delivery Ratio (PDR) for the different channel experiments.

1.3 Architecture

Figure 1.2 presents the architecture of system hereby proposed. A WiFi-enabled wearable device publishes user data to the system's broker. Then, the broker forwards the data to the subscriber which is a monitoring application with GUI. After receiving the data, the monitoring application stores the data in a database. Finally, to visualise the data, we use a data visualisation tool.

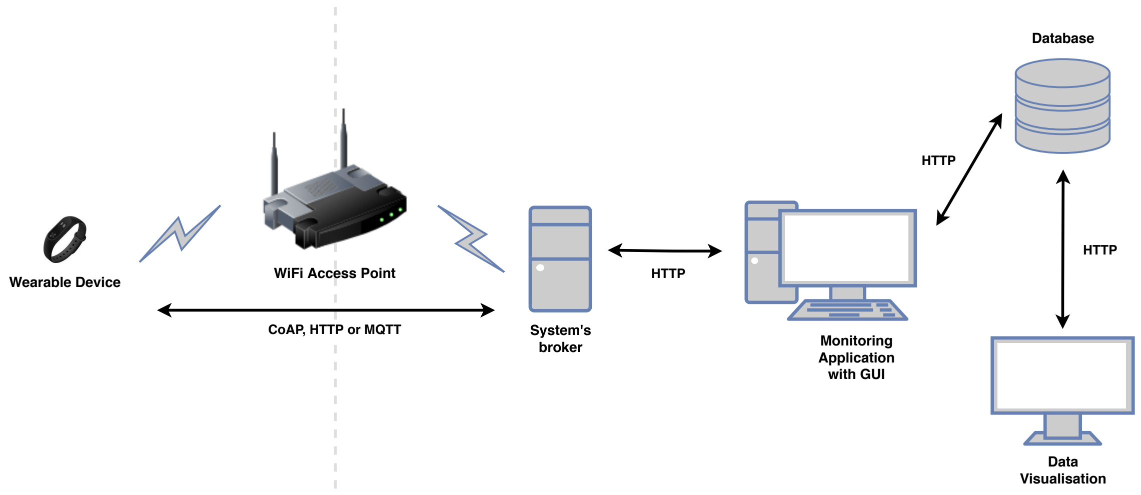


Figure 1.2: System architecture

1.4 Document Structure

This report is arranged in six chapters with the following content:

- **Chapter** 1 presents the context and motivation, objectives and system's architecture for this dissertation.
- **Chapter** 2 focus on areas-of-interest in developing an IoT system such as M2M and IoT applications development, low-power WiFi and related work to the dissertation, among others.
- **Chapter** 3 discusses the architecture of the IoT monitoring system as well as each component, giving an insight of its characteristics and function on the system.
- **Chapter** 4 discusses the system's implementation and operation.
- **Chapter** 5 presents performance results referred to E2E and PDR in the M2M communications.
- **Chapter** 6 discusses the project's final conclusions and future work.

Chapter 2

IoT Implementations on WiFi Nodes

This chapter covers four areas of interest for this dissertation. First, Section 2.1 reviews M2M and IoT applications development, in particular, middleware frameworks, open-source development frameworks, applications protocols and wireless communication technologies. Section 2.2 investigates the features of several embedded platforms with WiFi connectivity for IoT applications such as ESP32 and W600-PICO modules. Several options regarding time-series databases and data visualisation tools are discussed in Section 2.3. Finally, we conclude this chapter with related work to this dissertation from IoT applications with the implementation of the aforementioned standards, frameworks and ALPs in Section 2.4.1 and the innovative use of wrist-worn wearable devices in the sports domain in Section 2.4.2.

2.1 M2M and IoT Applications Development

As stressed earlier, IoT technologies and M2M have evolved rapidly in recent years. To respond to this growth, frameworks and M2M ALPs, as well as standards, were developed to provide solutions and improvements in this area. In this section, we will discuss several solutions regarding middleware frameworks, M2M frameworks and platforms (focusing on open-source products), ALPs and wireless communication technologies in M2M and IoT applications development.

2.1.1 Middleware Frameworks

This section presents several open-source middleware frameworks. It gives an overview of the main features and architecture.

2.1.1.1 AllJoyn

AllJoyn [4] is an open-source middleware framework supported by Open Connectivity Foundation (OCF) that follows the client-server cooperation model. Microsoft, Linux Foundation, Sony, Qualcomm are some of the members of the organization which support the framework. AllJoyn uses an object-oriented architecture to create its data models and offers bindings in Java, Objective-C, C++ and C [5]. Also, it is compatible with multiple operating systems, e.g., Windows, Linux, OS X, Android.

The AllJoyn framework architecture is divided into network architecture and software architecture. AllJoyn framework functions on the local network which allows device and apps discovery. There two components: AllJoyn Apps (Apps for short) and AllJoyn Routers (Routers for short). There are three common topologies, as illustrated in Figure 2.1 [6]:

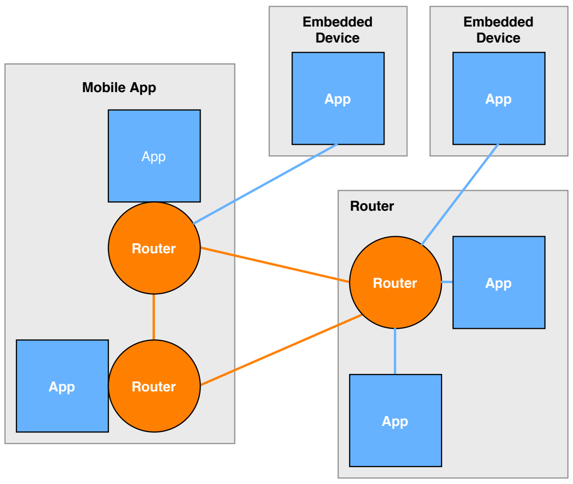


Figure 2.1: AllJoyn framework's network topologies

The software architecture details the different components in the Apps and Routers (refer to Figure 2.2). Apps contain the following components [6]:

- **Core Library**: Lower level APIs for interaction with AllJoyn network;
- **Service Framework Libraries**: implements common service like configuration, notifications and control panel;
- **App Code**: store the application logic.

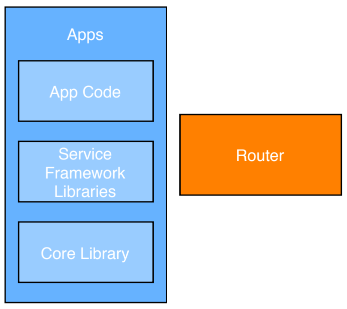


Figure 2.2: AllJoyn framework's network components

A Router can run as standalone device or is bundled with the AllJoyn Core Library.

Finally, the AllJoyn framework defines two variants. The Standard version is for non-embedded devices (e.g., Android and Linux) and the Thin version for resource-constrained devices (e.g., Arduino and Linux with limited memory).

2.1.1.2 FIWARE

FIMWARE [7] is an open-source framework developed to improve and accelerate the development of IoT solutions, and it was actively promoted by the European Community. It is supported by an independent Open Community whose members and contributors aim for a standard that is sustainable and easy to implement in order to build Smart Solutions in a faster, easier and cheaper way [8]. Also, FIWARE has an API called FIMWARE NGSI which uses the Next Generation Services Interface (NGSI) with the goal to unify the information's representation [9]. Furthermore, this framework enables the integration of components and offers the basis for the replication (portability) and interoperability for this field solutions. It is a RESTful API that provides an intuitive Graphic Users Interface (GUI), supports subscription/notification and many other features.

The main and only mandatory component of any platform based on FIWARE is the FIWARE Orion Context Broker Generic Enabler, providing several functions such as managing context information, performing updates and bringing access to context (refer to Figure 2.3) [10]. Moreover, there are complementary FIWARE components available with regards to IoT, API management and processing context information, among others [10]:

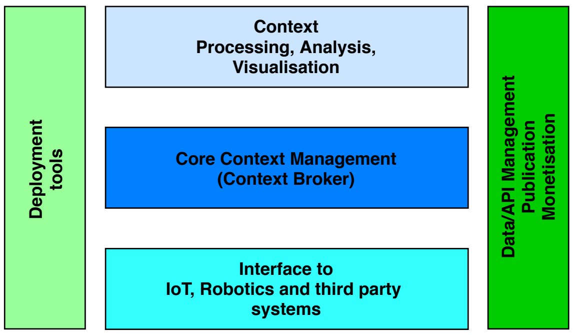

Deployment tools
Context Processing, Analysis, Visualisation
Core Context Management (Context Broker)
Interface to IoT, Robotics and third party systems
Data/API Management Publication Monetisation

Figure 2.3: General solution with a FIWARE platform

2.1.1.3 IoTivity

IoTivity [11] is an open-source and resource-oriented architecture framework supported by OCF which connects IoT devices, enabling and providing functions of discovery, connection and messaging among them [12]. M2M communications are supported through CoAP. Moreover, the subjective framework supports a connectivity abstraction layer. As a result, the API is available in multiple network technologies such as WiFi, Ethernet and Bluetooth and it is available for Linux, Windows, Android, Arduino, among others. Moreover, the API is available Java, C and C++ APIs. IoTivity framework contains three layers for different purposes: the service layer oversees device management, notification and Resource Container, among others; the base layer is in charge of messaging and discovery; and a cloud interface. Moreover, OCF developed a lightweighted platform called IoTivity-Lite for environments where resource management and energy efficiency are required.

Figure 2.4 illustrates the core of the IoTivity-Lite framework:

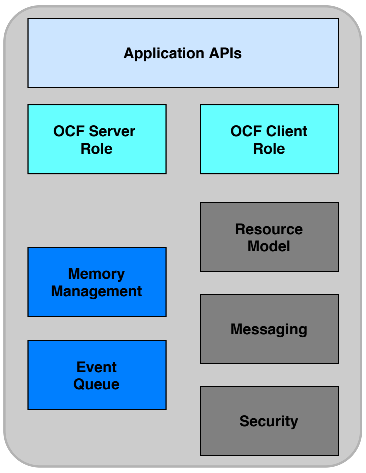


Figure 2.4: IoTivity-Lite core

2.1.1.4 OpenIoT

OpenIoT [13] is a joint effort from open-source contributors, funded by the European Union, to create large-scale intelligent IoT applications. This middleware was designed to manage and analyse data [5]. It is worth noticing that it can process every kind of data from almost any device, unlike other related solutions. Additionally, it runs on Linux, Windows and MAC OS operating systems.

The OpenIoT architecture is composed by three different logical planes which have different elements, as shown in Figure 2.5. These planes are the Utility/Application Plane, the Virtualized Plane and the Physical Plane [14].

It is worth noticing that the OpenIoT architecture is an instantiation of the reference architecture of the European Research on the Internet of Things (IERC) [15].

2.1.1.5 oneM2M

The oneM2M standard [16] is a global partnership project founded in 2012 by eight organisations including European Telecommunications Standards Institute (ESTI), Telecommunications Industry Association (TIA) and China Communications Standards Association (CCSA). The primary purpose is to address the lack of standardisation in M2M application, creating a single standard to resolve some of the issues associated with this field. As of now, oneM2M involves nearly 200 members and partners in which they have the possibility to test their solutions for the M2M market [17]. oneM2M functional architecture defines the following entities [18]:

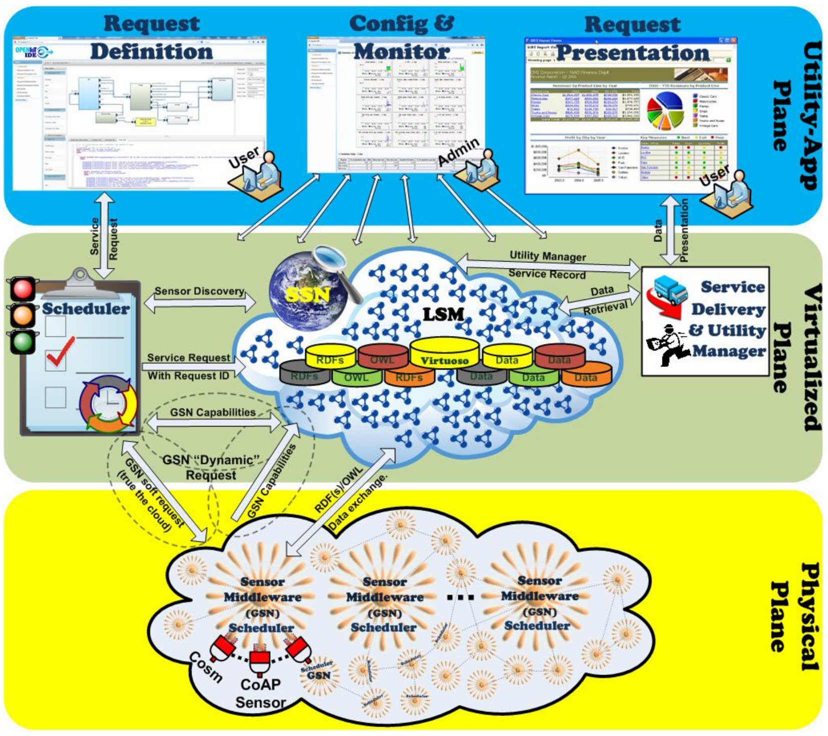


Figure 2.5: Open IoT logical planes [14]

- **Application Entity (AE)**: entity that represents M2M application service logic. An AE is an execution instance of an application service logic and has a unique identification (AE-ID). For instance, an application that monitors the HR is an AE.
- **Common Service Entity (CSE)**: entity that contains a set of functions that the AEs can use. Some of the services include data storage and group communication. Similarly to the AEs, a CSE has a unique CSE-ID;
- **Network Services Entity (NSE)**: is an entity that provides services to the CSE. The NSEs offer, for example, data transport services.

The communication between entities is achieved through reference points defined by the standard [18]:

- **Mca**: reference point for the communication between an AE and CSE;
- **Mcc**: reference point for the communication between two CSEs;
- **Mcc'**: reference point for the communication between two CSEs in Infrastructered Nodes (IN) but in different M2M Service Provider Domains;
- **Mcn**: reference point for the communication flows between a CSE and a NSE.

Finally, oneM2M standard defines a set of Nodes which contain CSEs and/or AEs. The types of Nodes in the system are distinguished between Nodes in the "Field Domain" - i.e, the domain

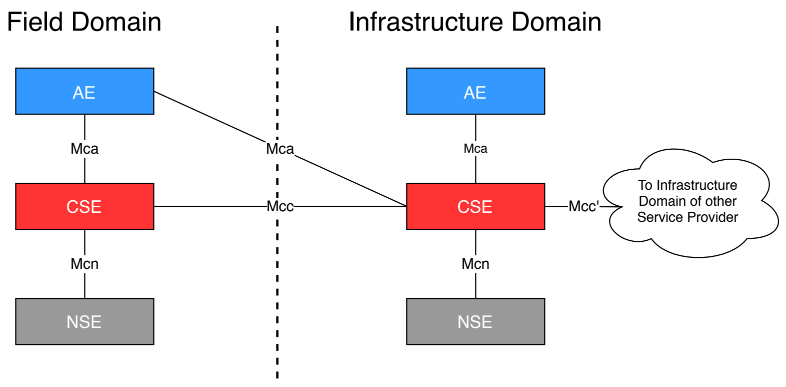


Figure 2.6: oneM2M functional architecture

where sensors, actuators, etc, are present - and "Infrastructured Domain" - i.e, applications and servers [18]. The oneM2M Node types are the following [18]:

- **Application Dedicated Node (ADN)**: a Node that is located in the Field Domain and contains at least one AE and does not have an CSE. Usually, these nodes are implemented in resource constraint devices that may not have processing and/or data storage resources. For instance, a simple sensor would be a ADN.
- **Application Service Node (ASN)**: a Node that is located in the Field Domain and contains one CSE and at least one AE. This Node can be implemented in resource constraint devices as well as more capable devices.
- **Middle Node (MN)**: a Node that is located in the Field Domain and contains one CSE and could contain AEs. Typically and MN would reside in a M2M Gateway.
- **Infrastructure Node (IN)**: a Node that is located in the Infrastructure Domain and contains one CSE and could contain AEs.
- **Non-oneM2M Node (NoDN)**: a Node that does not contain any oneM2M Entities, therefore implements non-oneM2M IoT applications. These nodes can be integrated in a oneM2M architecture by internetworking proxies.

Figure 2.7 shows the oneM2M layered model with the three layered models as well as the different nodes, entities and reference points:

The main objective of oneM2M standard are is to standardize M2M globally for the service layer as well as solve common IoT key problems. The main problems are listed below [19]:

- **Application Area**: oneM2M enables Application portability;
- **Data Interoperability**: oneM2M provides services towards the Application (Secure Communication, Device Management, etc) and enables Device portability (a Device can be connected to any Infrastructure solution);

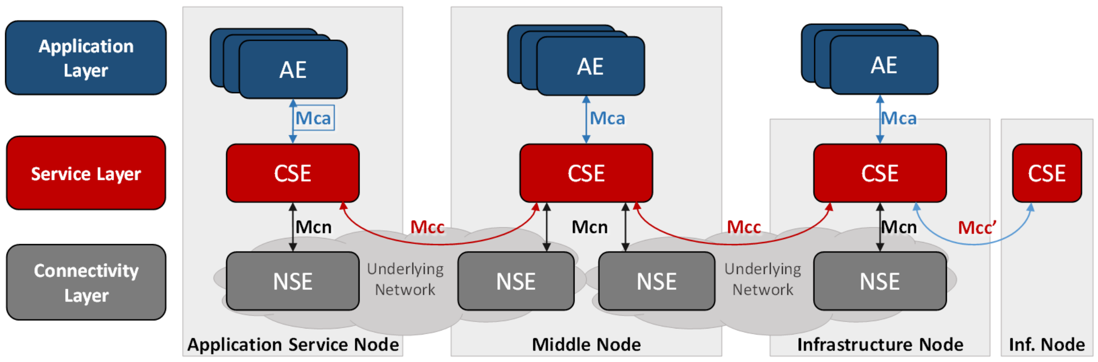


Figure 2.7: oneM2M layered model [18]

- **Connectivity**: oneM2M stores data in case of lack of connectivity and controls the devices usage of connectivity (When, how often communication happens).

2.1.2 Open-Source Development Frameworks

In Reference [5], the authors present the following open-source frameworks:

- The **Eclipse OM2M** [20] is an open-source platform, and implementation of the oneM2M and smartM2M standards, developed by the Eclipse Foundation based on the ETSI specification [21, 22, 23]. The project's software is plugin-driven which allows the integration of external plugins for additional ALPs and device management mechanisms, for example [24]. OM2M implements a RESTful API which exchanges Extensible Markup Language (XML) even through highly unreliable network conditions [24]. All framework's modules are organised in a service layer that offers resource discovery, device authentication, asynchronous and synchronous communications, along with other features [24, 5].

- **Sierra Wireless Legato** [25] is a framework which offers a set of APIs for network services, cloud and database accessing services [5]. The programming languages of the API are C and C++ [5]. The Legato Application Framework, illustrated in Figure 2.8, is a Runtime Environment which enables the system to run and monitor applications as well as sending and receiving data from other sources [26].

- **OpenMTC** [27] is a framework offering M2M services being a reference implementation of the oneM2M standard. OpenMTC consists of a gateway and network layers which are service capable [5]. The framework supports RESTful architecture and follows client and server communication [5]. Also, information is exchanged between the two aforementioned layers. This framework supports HTTP(S) and MQTT bindings, TLS-based security, data storage (in-memory), among others. Figure 2.9 illustrates the framework's architecture:

  Finaly, the OpenMTC software is written in Python and can be supported in multiple hardware platforms such as x86 and ARM [27].

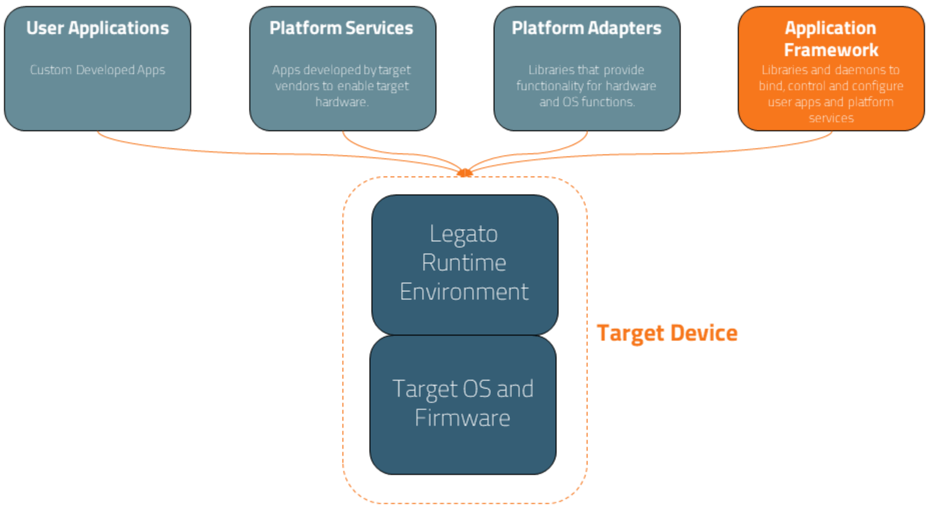

User Applications
Custom Developed Apps
Platform Services
Apps developed by target vendors to enable target hardware.
Platform Adapters
Libraries that provide functionality for hardware and OS functions.
Application Framework
Libraries and daemons to bind, control and configure user apps and platform services
Legato Runtime Environment
Target OS and Firmware
Target Device

Figure 2.8: Legato application framework [26]

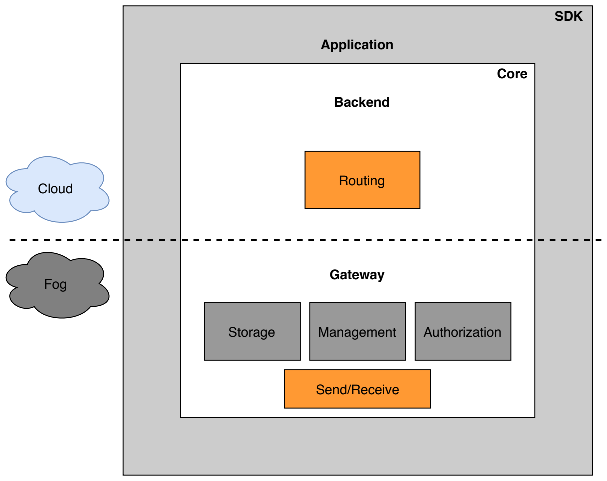


Figure 2.9: OpenMTC framework's architecture

2.1.3 Application Layer Protocols

The ALP define how information is exchanged between devices connected to a network, allowing devices with different resources or software communicate with each other. In particular, IoT technologies have devices with limited resources, thus the choice of a ALP is important because it affects communication efficiency, for example. In this Section, we present the most relevant aspects of protocols that are adequate to IoT technologies, namely the AQMP, CoAP, HTTP, MQTT and XMPP.

2.1.3.1 Advanced Queuing Message Protocol

Advanced Queuing Message Protocol (AQMP) is a M2M protocol designed with the focus for interoperability between multiple manufacturers, reliability, security and provisioning [28]. The protocol supports publish/subscribe and request/response cooperation models [29]. As the protocol's name implies, it offers various features regarding messaging such as publish-subscribe topic-based messaging, reliable queuing, flexible routing and transactions [28]. AQMP uses Transmission Control Protocol (TCP) as its transport protocol. Adding to the reliability inherent to TCP connections, AQMP provides three QoS levels known by *at-most-once*, *at-least-once* and *exactly-once* [30].

2.1.3.2 CoAP

CoAP is a lightweight M2M protocol developed by the Internet Engineering Task Force (IETF) for resource and network constrained devices [5, 31, 32]. Therefore, CoAP was designed to replace HTTP in resource-constrained devices [31] and uses Universal Resource Identifiers (URI) instead of topics. The protocol is based in a RESTful architecture style [33] and uses User Datagram Protocol (UDP) as its transport layer protocol. As a result, CoAP possesses a request/reply structure, low overhead and unreliability regarding message delivery. Due to its connection-less communication, two protocol variants are defined regarding QoS. CoAP Confirmable messages are acknowledged by the receiver as opposed to CoAP Non-confirmable, wherein the sender does not receive an acknowledge message by the receiver.

To achieve a publish/subscribe cooperation model, which is desirable for a monitoring system with resource constrained devices, the protocol offers a feature to receive resource information asynchronously [34]. With the OBSERVE option in a GET request, the client notifies the server to send resource updates whenever the resource information changes.

2.1.3.3 HTTP

HTTP is a standard developed by the IETF and a published standard since 1997 being globally used as web messaging protocol [29]. This protocol supports a request/response RESTful architecture and, analogous to CoAP, uses URI [29]. HTTP uses TCP as its transport layer protocol [29]. Therefore, the communication between the client and server is reliable. Unlike previous protocols, it does not offer nor define QoS levels.

Moreover, the communications between the client and the server start when the client makes a request. There are four types of client requests:

- **DELETE**: Removes an existing resource.
- **GET**: Get information about a resource or resources that already exist;
- **POST**: Creates a new resource;
- **PUT**: Updates the information of an existing resource;

Thus, there is an inherent abstraction on the client-side of how the server processes the requests which enable a stateless approach because the server doesn't need to maintain the client's state after a response to a request.

2.1.3.4 MQTT

MQTT is a lightweight and one of the oldest M2M communication protocols developed by IBM and introduced in 1999 [31, 32, 29]. The protocol follows a topic-based publish-subscribe architecture designed and optimised for high-latency and constrained network [35, 36]. Analogous to HTTP, MQTT uses TCP as its transport layer protocol.

As stressed earlier, the protocol follows a topic-based publish-subscribe model which means that each message is associated with a topic. The publisher entity sends the messages to a broker, which acts as an intermediary and forwards the messages to the entities that subscribed to the message's topics.

MQTT offers three different levels of QoS regarding message delivery [37]:

- **QoS 0** (at most once delivery): This offers the inherent message delivery reliability of the TCP protocol. In this level, the message arrives at the receiver or does not arrive at all. The sender does not retry to send the message and there is no response message by the receiver;
- **QoS 1** (at least once delivery): This QoS level ensures that the message is delivered at least once to the receiving entity. The receiver has to send a confirmation message to the sender to inform that the message was delivered. If the sender doesn't receive a confirmation message after a timeout, the message is resent;
- **QoS 2** (exactly once delivery): This QoS level ensures that there are no duplicate messages or message loss. Both entities send confirmation messages to ensure that the message is delivered exactly once to the receiver entity.

As expected, communication reliability increases with the levels of QoS. However, there is more latency and bandwidth consumption associated with higher QoS levels.

2.1.3.5 Extensible Messaging and Presence Protocol

Extensible Messaging and Presence Protocol (XMPP) is a protocol based on extensible markup language (XML) developed by IETF. Analogous to AQMP and MQTT, it is designed for message-oriented middleware [5]. XMPP is based on TCP protocol with XML Stanzas and supports real-time communication between a server and a client. The protocol follows both publisher/subscriber and request/response patterns [31].

2.1.3.6 Protocol's Comparison and Evaluation

In the following table, a comparative analysis between the previously mentioned protocols is presented:

Table 2.1: Comparative analysis of application protocols in IoT systems

Protocol	Abstraction	Architecture	Quality of Service	Transport Protocol
AQMP	Publish/Subscribe Request/Response	Client/Broker Client/Server	Settle Format Unsettle Format	TCP
CoAP	Publish/Subscribe Request/Response	Client/Broker Client/Server	Confirmable Non-confirmable	UDP
HTTP	Request/Response	Client/Server	N/A	TCP
MQTT	Publish/Subscribe	Client/Broker	QoS0, QoS1, QoS2	TCP
XMPP	Publish/Subscribe Request/Response	Client/Server	N/A	TCP

During the last decade, due to the growth of M2M communications and IoT applications, there has been intensive research and study of M2M communication protocols performance in different applications scenarios with different technologies and network conditions, for example. Therefore, Table 2.2 shows relevant studies regarding the protocols previously mentioned, among others. It is worth noticing that the table follows the structure presented in Reference [32].

Table 2.2: Comparison study about ALPs in previous works

Authors	Protocols	Metrics	Purpose	Results
Bandyopadhyay et al. [35] (2013)	CoAP, MQTT	Power consumption by increasing payload size and changing the conditions for packet loss regarding payload size	Study protocol's performance regarding energy efficiency using Constrained Gateway Devices	CoAP is the most efficient in terms of energy comsuption and bandwidth
Fysarakis et al. [36] (2016)	CoAP, DPWS, MQTT	Client response time to a request, Average CPU Load and Average Memory Utilisation	Evaluate the protocols in the same testbed to extract conclusions concerning performance	DPWS showed the worst results in two categories (client response time and memory utilisation), followed by MQTT (CPU load).

Chen et al. [31] (2016)	CoAP, Custom UDP, DDS, MQTT	Bandwidth consumption, latency and packet loss	Conduct a study to illustrate how protocols function how protocols function under a constrained, low quality wireless networks	TCP-based protocols are more reliable and have more latency experienced than UDP-based protocols
Kayal et al. [38] (2017)	CoAP, MQTT, XMPP, WebSocket	Response time by varying the traffic load on the network	Measures protocols' performance in constrained devices to ensure efficiency	CoAP performs better than other protocols for higher server utilisation. XMPP performs better than others at lower server utilisation
Hedi et al. [39] (2017)	CoAP, MQTT	Bytes sent and time needed to receive them	Evaluate performance and compare the protocols in different scenarios	Concludes that CoAP is a good choice where real-time performance and latency are not requirements

Naik [29] (2017)	AMQP, CoAP, HTTP, MQTT	Message Size vs Message Overhead, Power Consumption vs Resource Requirement, among others	In-depth and relative analysis of the protocols to gain insight into their strengths and limitations	HTTP is the first (i.e, with highest) in message size, message overhead, power consumption and latency while CoAP is the lowest. MQTT is the highest in reliability and AQMP is highest in security and provisioning, for example.
Pohl et al. [40] (2018)	AMQP, MQTT, XMPP	Latency, Throughput, Bandwidth Usage, Reliability and Energy Consumption	Measures protocols' performances regarding variable latency and Packet-Loss-Rate (PLR)	MQTT performs best regarding the characteristics bandwidth usage, reliability, latency and throughput. AMQP performs worse than MQTT comparing the bandwidth usage and throughput. In contrast, XMPP has the worst values in all categories

Pavelic et al. [41] (2018)	CoAP, HTTP, MQTT	Measuring energy and power Consumption sending data and standby	Measuring protocols' performance by sending data and bring in a standby position regarding energy and power consumption	CoAP and MQTT preformed significantly better than HTTP. CoAP and MQTT consume almost equal amounts of energy for sending data
Çorak et al. [32] (2018)	CoAP, MQTT, XMPP	Measure packet creation time and packet transmission time	Collect real-time environmental data with a real-world testbed	XMPP is worse than other protocols in both metrics and MQTT and CoAP perform almost equally

2.1.4 Wireless Communication Technologies

In recent years, wireless communication technologies have emerged to respond to M2M communications requirements. Some technologies allow long-range communications and high data transmission rates, while others aim to short-range M2M communications with low data transmission rates. In this work, we focus and give an overview of short-range wireless communication technologies because these technologies are the most relevant for an online monitoring system.

2.1.4.1 Bluetooth

Bluetooth, standardised as IEEE 802.15.1, is a wireless communication technology developed as an alternative to wire-based communication technologies. Since the first consumer Bluetooth device launch in 1999, the standard is continuously evolving to adapt to the increasing technology requirements, playing a major role in M2M communications nowadays.

Bluetooth devices operate in the 2.4GHz ISM spectrum band and use 79 of its channels. As of now, the standard defines four device classes differentiating transmission power and range. Regarding architecture, Bluetooth follows a master/slave architecture, and each master can connect at the same time to seven slaves devices at most in a piconet network.

Although several online monitoring systems implementations are using Bluetooth has its short-range M2M communications technology, the standard allows networks with few devices, which makes it unsuitable for more ambitious monitoring systems in terms of scale. Moreover, the high startup times to connect to a new device and its high energy consumption are not adequate for this use case. The introduction of the low-energy consumption version, Bluetooth Low Energy [42], makes the standard more suitable for this type of systems. Finally, a Bluetooth-based system normally needs a GW device (p.e., a Smartphone) to connect to the Internet, which typically uses WiFi at its border.

2.1.4.2 Zigbee

ZigBee is a low-power communication protocol for personal area networks with small and low-power devices such as home automation, data collection for medical personnel and other small projects. Zigbee operates in 2.4GHz, which can be an issue due to its overlap with WiFi and other technologies. Additionally, the standard offers low data transmission rates and limited support to QoS. The connection to the Internet is normally achieved through a GW node.

2.1.4.3 WiFi

The IEEE 802.11 protocol, more commonly known as WiFi, allows the creation of a Wireless Local Area Network and provides Internet access. The technology offers secure, reliable and fast wireless connectivity between devices. WiFi is present in the various scenarios in today’s society, such as office spaces, commercial areas and homes.

The network operates in 5GHz or Super High Frequency (SHF) and 2.4GHz ISM spectrum bands while providing Internet connectivity through APs.

In recent years, with the emergence of small WiFi-enabled low-power devices, more monitoring systems use this technology as its M2M communication protocol. Moreover, the standard offers power management features which makes it suitable for systems with low-power requirements. As opposed to Bluetooth, systems with WiFi technology does not need a GW device for an Internet connection.

2.2 Low-Power WiFi Nodes

Nowadays, there is a vast choice regarding embedded platforms with WiFi connectivity [43]. Although multiple devices are available in the market, there are discrepancies concerning wireless communication interfaces (e.g, Bluetooth, WiFi and Zigbee), computational power, cost and energy consumption as well as physical footprint [43]. The ESP32 and the W600-PICO are some of the modules available on the market with WiFi connectivity which can be easily applied to IoT applications [44, 45]. In the following sections we cover the main features of these modules that are particularly interesting to IoT applications.

2.2.1 ESP32 Module

ESP32 is an ultra-low-power solution designed for mobile, wearable devices and, more importantly, for IoT applications [44]. The module contains the following features that make it a highly-integrated solution for IoT applications [44]:

- Standard IEEE 802.11 b/g/n and IEEE 802.11 n (2.4 GHz, up to 150 Mbps) compliance with on-board antenna;
- WPA/WPA2 security protocols;
- Xtensa® single-/dual-core 32-bit LX6 microprocessors(s), up to 600 MIPS;
- 520kB internal SRAM and 16MB external QSPI flash;
- Five built-in power modes supported;
- 25x18 mm;
- Low-cost device, when compared to other WiFi modules.

Concerning the 32-bit processor, it provides computing power enabling the following features, which eases the development of IoT application [43]:

- Real-Time Operating System (RTOS);
- Infrastructure station, SoftAP and Promiscuous modes [44];
- Complete TCP/IP protocol stack.

2.2.2 W600-PICO Module

W600-PICO module is a device with an WiFi connectivity that can be easily applied to IoT applications such as smart appliances, smart homes and healthcare [45]. This device has the following features [45]:

- Standard IEE 802.11 b/g/n/e/i/d/k/r/s/w compliance with on-board antenna;
- WPA/WPA2/WPS security protocols;
- Supports WiFi WMM/WMM-PS;
- Arm® Cortex M-3 32-bit processor with 80 MHz operating frequency;
- 288 KB internal RAM and 1MB/2MB internal flash;
- 33x20.3 mm;

Similar to the ESP32 module, the W600-PICO 32-bit processor provides computing power enabling the following feature:

- Real-Time Operating System;
- Supports AP and STA modes;

Regarding power management, W600-PICO supports PS-Poll and U-APSD defined by the IEEE 802.11 standard [45]. It is worth noticing when in standby, W600-PICO power consumption is less than 10 μA [45].

2.2.3 WiFi Nodes Comparison

Table 2.3 presents the main features of the ESP32 and W600-PICO modules that ease the development of IoT applications.

Table 2.3: Main features of the ESP32 and W600-PICO modules

Feature	ESP32	W600-PICO
WiFi	Standard IEEE 802.11 b/g/n and IEEE 802.11 n (2.4 GHz, up to 150 Mbps) compliance with on-board antenna	Standard IEEE 802.11 b/g/n/e/i/d/k/r/s/w compliance with on-board antenna
Security Protocols	WPA/WPA2 security protocols	WPA/WPA2/WPS security protocols
Processors	Xtensa® single-/dual-core 32-bit LX6 microprocessors(s)	Arm® Cortex M-3 32-bit processor with 80 MHz operating frequency
Memory	520kB internal SRAM and 16MB external QSPI flash	288 KB internal RAM and 1MB/2MB internal flash
Power Management	Four built-in sleep modes supported with minimum current consumption of 10 μA	PS-Poll and U-APSD defined by the IEEE 802.11 standard. In standby the power consumption is less than 10 μA
Dimensions	25x18 mm	33x20.3 mm

2.3 Time Series Databases and Data Visualization Tools

An IoT monitoring system can collect and store thousands or even millions of data instances over a short period. Therefore, data processing, management and visualisation are key aspects of an IoT monitoring system. In this section, we present three alternatives that accomplish the former aspects: InfluxDB and Chronograf, Prometheus and Grafana and, finally, Thinger.io.

2.3.1 InfluxDB and Chronograf

InfluxDB [46] is an open-source time-series database built to handle high write and query loads. The database is one of the four components that form the TICK (Telegraf, InfluxDB, Chronograf, Kapacitor) stack. InfluxDB is a database for use cases that involve large amounts of timestamped data such as IoT sensor data monitoring and real-time analytics [47]. It currently supports several key features that are useful and important timestamped data-centred system [47]:

- Custom designed datastore for time series data;
- Written exclusively in Go and compiles into a single binary file without external dependencies;
- Offers high performing write and query HTTP APIs;

- Developed to query aggregated data with an SQL-like language;
- Data retention policies.

Chronograf [48] is the user interface for the InfluxDB 1.x platform with the purpose to allow the users to see data stored in the database with ease. The software includes templates and libraries to build dashboards with real-time visualisations of data. Together with Kapacitor [49], it is possible to create different types of alerts for detecting anomalies in the stored data and see the alerts history in the Chronograf user interface. Figure 2.10 illustrates the TICK stack and its behaviour model:

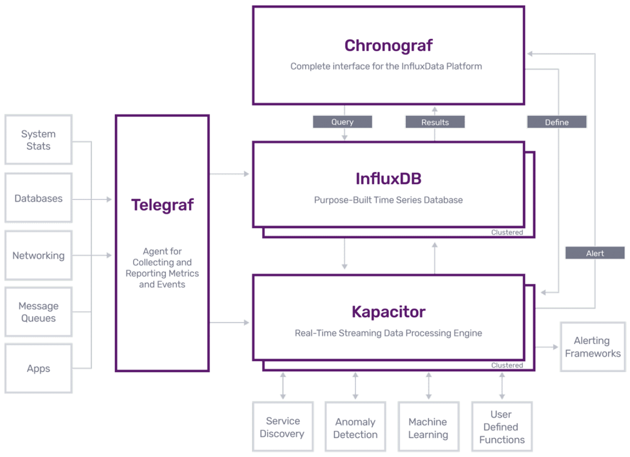


Figure 2.10: TICK stack [46]

2.3.2 Prometheus and Grafana

Prometheus [50] is an open-source database designed for systems that rely on timestamped data created SoundCloud. Prometheus main features are the following [51]:

- Does not rely on distributed storage, single server nodes are autonomous;
- Data collection via HTTP pull model;
- Multiples modes of graphic visualisation and support;
- A flexible query language, PromQL, to query aggregate data;
- Written primarily in Go.

Figure 2.11 illustrates the Prometheus architecture and some of its primary components, many of which can be replaced by other tools:

Grafana [52] is the tool for real-time data visualisation stored in Prometheus, analogous to what Chronograf is to InfluxDB. Grafana offers templates and libraries to built custom dashboards as well as an alert management system [53].

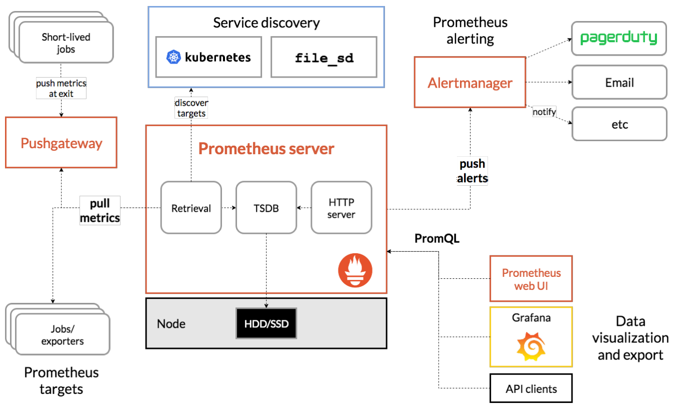


Figure 2.11: Prometheus architecture [51]

2.3.3 Thinger.io

Thinger.io [54] is an open-source cloud platform specially designed for IoT architectures. The platform consists of a Backend (which is the IoT server) and a web-based Frontend for both computer and smartphone to manage all the available features. Thinger.io has the following key features [55]:

- Hardware agnostic, which means that any product, independent of the manufacturer, is easily deployable and integrated;
- Efficient, efficient and affordable ways to store device data as well as real-time data aggregation;
- Real-time data visualisation through dashboards;
- Alert management system.

Figure 2.12 shows the main components of a Thinger.io ecosystem:

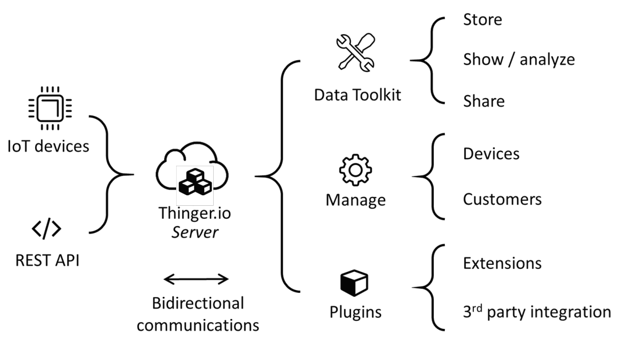


Figure 2.12: Thinger.io architecture [55]

2.4 Related Work

In this section we show some works that are specifically related to our objectives. First, in Section 2.4.1, we discuss IoT applications that implement the standards and frameworks referred previously. Then, in Section 2.4.2, we present a study of innovative uses of wrist-worn wearable devices in the sports domain.

2.4.1 IoT Implementations

Akasiadis et al. [24] implement an IoT framework with open-source frameworks that supports multiple ALPs intending to develop a unified solution by designing a system that is easily deployable and reusable on various application domains. The authors present an IoT platform with seamless interconnections between services and datastreams, support for multiple ALPs among other features. The platform supports MQTT, AMQP, WebSockets, CoAP and REST HTTP which are five of the most used ALPs. Moreover, it is based in open-source frameworks that are interconnected and provide support for the oneM2M standard (with the Eclipse OM2M framework), semantic descriptions that can be used for service discoverability and reasoning, and authentication/authorization (AuthN/AuthZ) procedures for services and datastream access and sharing. An instance of the SYNAISTHISI platform as a dockerized container is set up to integrate all the functionalities described.

Pereira et al. [3] develop an open IoT for continuous monitoring for emergency wards. The authors propose an architecture to monitor patients' physiological parameters, fully integrated with Internet from the wearable sensors to the monitoring system. At the lower end, the authors use low-cost and low-power WiFi-enabled wearable physiological sensors based on the ESP8266 module that connects directly to the Internet infrastructure avoiding the use of a GW device. Additionally, the architecture is based on the oneM2M standard. Finally, at the upper level, the architecture relies on the openEHR framework for storage, monitoring and data semantics.

Pereira et al. [23] compare the FIWARE and oneM2M middleware platforms in a publish-subscribe architecture. The authors implement CoAP, HTTP and MQTT to obtain a quantitative analysis for both middleware platforms regarding publish and subscribe times and goodput and publish/subscribe sizes.

Aghenta et al. [56] develop a Supervisory Control And Data Acquisition (SCADA) based, low-cost and open-source IoT system wherein current and voltage sensors acquire data while an ESP32 microcontroller receives, processes and sends the data to Thinger.IO local server (hosted in a Raspberry Pi microcontroller) which stores, monitors and controls the data. The authors present two configurations regarding the connection between the hardware components. The authors propose two scenarios: the user can access the data over the Internet or just when connected to a local WiFi. For the first, the Raspberry Pi has to be connected to the network via Ethernet cable while in the second scenario, the Raspberry Pi is connected to one of the LAN ports of the local WiFi router.

Yadav et al. [57] present an IoT application for a healthcare system scenario with monitoring the heart rate being the use case. In particular, the authors implement an IoTivity-based architecture for a heartbeat sensor application. The heartbeat sensor is connected to a smart thing server and use CoAP as the ALP to communicate with data management blocks.

2.4.2 Wrist-Worn Wearable Devices in the Sports Domain

Santos-Gago et al. [58] conducts a study in which the authors present innovative proposals on the use of wrist-worn wearable devices in the sports domain. This study was oriented to the use of wearable devices for monitoring of athletes behaviour in activities not supported by the vendors, identifying specific types of movement or actions in specific sports and preventing injuries. Table 2.4 only shows the works that used online monitoring of physiological parameters for their final purpose.

Table 2.4: Innovative use of wearable technology for monitoring physiological parameters

Reference	**Use**	**Device**	**Sport**
Walker et al. [59] (2016)	Estimate energy expenditure duting training and competition. The product aims to improve the physical performance of the players	Only commercial wearables were used to monitor oxygen consumption: SenseWear Armband (Model MF-SW) to estimate energy expenditure; and MiniMax4.0 (Scoresby Australia)	Australian football
Kos and Kramberger [60] (2017)	Use wearables to estimate the type of shot made by detecting the impact of the racket against the ball	Ad-hoc wearable device that includes 3D gyroscope, 3D accelerometer (±16 g), a heart rate sensor, and temperature sensor	Tennis
Parak et al. [61] (2017)	Commercial wearable with an embedded optical heart rate sensor that estimates heart rate, energy expenditure and maximal oxygen outtake (VO_2 max) while running	Commercial wearables for running practice. To estimate heart rate, the PulsOn was used and the Samsung Galaxy S3 smarthphone was used for geolocation	Running

Enomoto et al. [62] (2018)	System developed to monitor lactic acid secretion on the surface of the skin. Sensor monitors lactic acid secretions while exercising	Ad-hoc device combining a biosensor using LOD and osmium wired HRP (Os-HRP) reaction system with a microflow-cell	Generic
Soltani et al. [63] (2019)	Estimate gait speed during outdoor exercise (walking and running) using a energy-efficient wearable device	Commercial wearables for running practices: wrist-worn inertial sensors (Physilog®IV, GaitUp, CH), and a head-worn Global Navigation Satellite System (GNSS) device as a location reference	Running

2.5 Summary

As shown in Section 2.1, there are multiple options available regarding middleware frameworks, open-source frameworks, ALPs and wireless communication technologies. In particular, multiple ALPs were developed with the growth of IoT systems and with different characteristics. For example, Reference [35] shows that CoAP is the most efficient in terms of energy consumption and bandwitdh and Reference [31] concludes that TCP-based protocols are reliable, but have more latency than UDP-based protocols, as expected. Section 2.2 showed that the ESP32 and W600-PICO modules are capable devices for IoT applications. We present time-series databases and visualisation tools in Section 2.3 and concluded this Chapter with work that is specially related to this dissertation in Section 2.4. In particular, we presented IoT architectures that implement one of the middleware frameworks and/or open-source frameworks and then we showed a study about innovative use of wrist-worn devices where there exist no online monitoring systems yet, using ESP32 modules exploiting direct Internet connection via WiFi.

Chapter 3

Components of the IoT Monitoring System

This chapter introduces the proposed architecture of the IoT system focusing on its main components that envision to satisfy the requirements for remote health monitoring. For each component, we give a description and insight into each component's function in the system as well as a detailed architecture with the oneM2M standard entities. At first, in Section 3.1, we detail the Eclipse OM2M project because it is the central element of the system. Section 3.2 investigates the system wearable, which comprises an ESP32 module, a MAX30102 PPG module and a Lithium-ion Polymer (LiPo) battery. The following section presents and describes the monitoring application with GUI. We conclude the chapter with a description of InfluxDB and Chronograf, which are for data storage, management and visualisation, in Section 3.4 and 3.5, respectively.

3.1 OM2M

Eclipse OM2M [20], or simply OM2M, is an open-source platform that implements the oneM2M standard. The central element of the architecture is a Common Service Entity (CSE), which provides important features to an IoT system such as device management, device, internetworking, security and notification [20].

OM2M relies on a RESTful API for creating and managing M2M resources, including primitive procedures to enable applications registration, containers management, synchronous and asynchronous communications, among others [64]. The platform is a Java implementation running on top of a plugin-based software [64].

To support the oneM2M standard, OM2M's API handles the following primary resource types [64]:

- **CseBase**: defines the hosting CSE and is the root for all system's resources;
- **remoteCse**: stores information related remote M2M CSEs residing on other M2M machines;

- **AE**: stores information about the AE;
- **Container**: enables data exchange between applications and CSEs;
- **AccessControlPolicies**: manages permissions and permissions holders;
- **Group**: enables an issuer to send one request to a set of receivers instead of sending requests one by one;
- **Subscription**: stores information related to subscriptions for resources and allows subscribers to receive a notification when an event happens;

The platform follows a publish-subscribe architecture which makes it a good choice for a monitoring system, with the CseBase functioning as the system's broker.

Finally, the platform offers plugins to support ALPs bindings such as CoAP and HTTP. Although the OM2M offers MQTT binding, it does not include an MQTT broker, thus the need an external MQTT broker. The Eclipse Mosquitto [65] is one of the most common choices because the OM2M broker uses the Eclipse Paho [66] library as an MQTT Client for the MQTT binding.

Figure 3.1 revisits the architecture presented in Figure 1.2 but with the oneM2M standard entities. The system comprehends two domains, Infrastructure Domain and Field Domain. Regarding the Field Domain, it only was entity referred as ADN-AE, which represents all the logic behind the wearable. On the other hand, the Infrastructure Domain retain an IN-CSE entity and one IN-AE entity. The logic behind the IN-CSE entity is the OM2M platform that supports CoAP, MQTT and HTTP protocols. The IN-AE entity represent the monitoring application. InfluxDB and Chronograf are not entities defined on the standard.

In this scenario, both the ADN-AE and IN-AE publish and subscribe data from the IN-CSE. The communication between the ADN-AE and the IN-CSE is via CoAP, HTTP or MQTT. However, HTTP is utilised for the IN-CSE and IN-AE communication as well as for the communication between the IN-AE and the InfluxDB and Chronograf.

3.2 Wearable

The wearable proposed in this system includes an ESP32 module, a MAX30102 module, or simply ESP32 and MAX30102, and a LiPo battery. The device has the purpose to acquire and send to a broker, the physiological parameters from the users as well as the device's battery percentage via the WiFi infrastructure.

The LiPo battery is a rechargeable battery used in applications where weight and overall size are requirements, such as a wearable device. This battery powers both the ESP32 and the MAX30102. Chapter 4 gives a description on how the components are connected.

In Section 3.2.1, we present the key features and characteristics, power modes configurations and the development framework used. We conclude this section with a description of the MAX30102, in Section 3.2.2, including the primary registers and FIFO data structure. It is worth

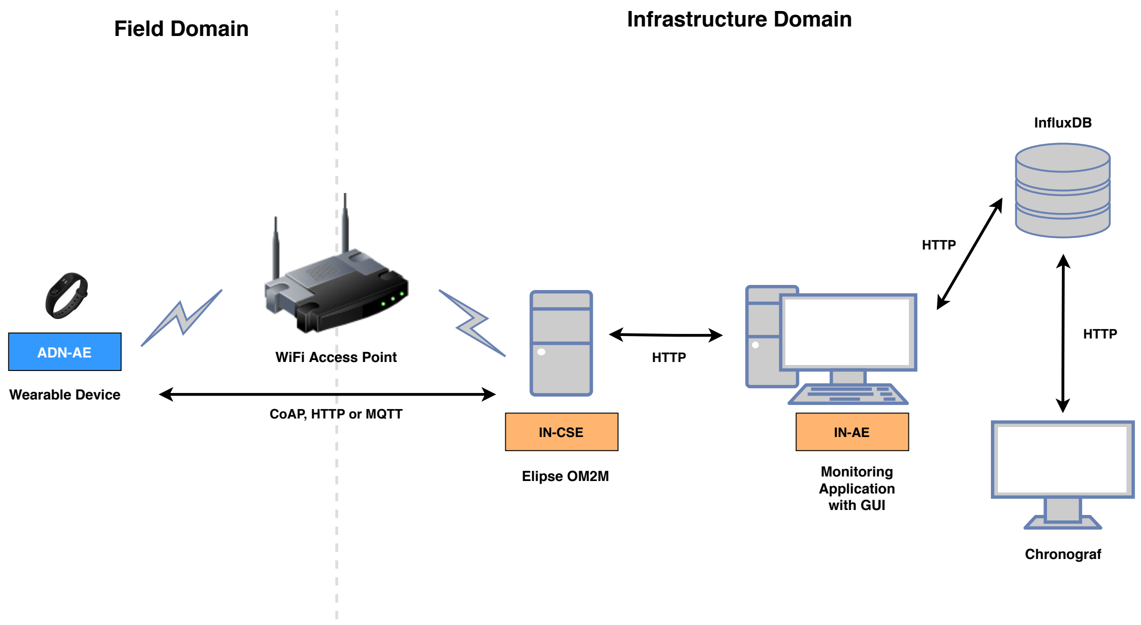


Figure 3.1: System architecture with oneM2M standard entities

noticing that we present the connection between the ESP32 and MAX30102 and, consequently, data processing in Chapter 4.

3.2.1 ESP32 Module

The ESP32 WiFi module, developed by Espressif Systems and released in 2016, is the successor to the ESP8266 WiFi module. It offers a System-on-Chip (SoC) solution to meet the requirements of M2M communications and IoT applications due to its built-in power management mechanism associated with the WiFi technology. Figure 3.2 presents a high-level block diagram of the ESP32 module.

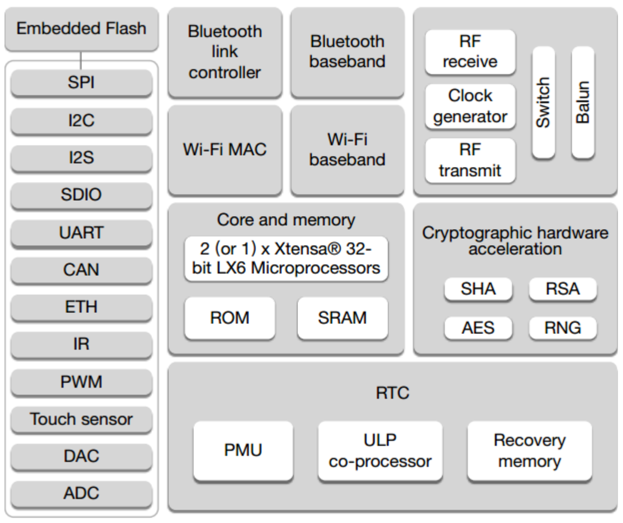


Figure 3.2: ESP32 module's high-level block diagram [44]

There are five variations of the ESP32 chip, which primarily differentiate with the number of CPU cores (one or two), MIPS value and whether there is embedded flash memory or not:

- ESP32-DOWDQ6;
- ESP32-D0WD;
- ESP32-D2WD;
- ESP32-S0WD;
- ESP32-PICO-D4.

For this project, we chose ESP32-D2WD chip because it has a dual-core CPU processor with 600 MIPS and a 16 MB embedded flash memory. This particular chip offers several features that make it highly integrable with IoT applications [44]:

- Small size (25 x 18 mm), ideal for wearable devices, for example;
- Supports IEEE 802.11 b/g/n and IEEE 802.11 n (2.4 GHz, up to 150 Mbps) compliance with on-board antenna, which enables sensing applications with an Internet connection without a GW device;
- Support WPA/WPA2 security protocols for a secure WiFi connection;
- Power management features such as multiples power modes and dynamic power scaling;
- Two 32-bit processors, with low-power consumption and with a maximum operating frequency of 240 MHz (Xtensa® dual-core 32-bit LX6 microprocessors(s)).
- Supports a real-time operating system (FreeRTOS);
- Offers three WiFi operating modes: Infrastructure Station, SoftAP and Promiscuous;
- Full TCP/IP stack implementation;
- Integrates 520kB internal SRAM and 16MB external QSPI flash;
- Low-cost device, when compared to other WiFi modules.

The Espressif IoT Development Framework, also know as ESP-IDF, is the official SDK for the ESP32 family series. It has support for Windows, Linux and Mac OS. ESP-IDF provides the developers with a FreeRTOS-based API [67], written in C, to help them do develop applications with ease. Furthermore, Espressif System provides up-to-date API documentation [68] and practical examples of the different API functionalities.

We chose the ESP32 as the wearable's microcontroller due to its characteristics that make it a easily deployable module for IoT application, namely a 32-bit dual-core microprocessor, support for five built-in power modes and RTOS, and a complete TCP/IP protocol stack.

3.2.1.1 Low-power Management

As stressed earlier, this module offers five power modes which enable the CPU, the WiFi interface and other peripherals to shutdown for inactivity periods [44].

The *Active* mode is the default mode where the CPU, WiFi interface and other peripherals are enabled. Thus, it is not ideal to use this mode if the application requires low-power consumption.

In *Modem-sleep* mode, the CPU is operational and the clock frequency is configurable. The WiFi circuit is shutdown although the association to network AP is maintained, thus avoiding the need to reconnect upon waking and the respective high latency [43]. Additionally, Espressif defines two variants for the *Modem-sleep*, *Minimum Modem sleep* and *Maximum Modem sleep*, only if the module works in station mode [69]:

- *Minimum Modem sleep*: the device wakes up every Delivery Traffic Indication Message (DTIM) to receive a beacon. Broadcast data will not be lost because it transmitted after DTIM. However, if the DTIM is short, there is not that much power saving;

- *Maximum Modem sleep*: the device wakes up every beacon listen interval. Broadcast data can be lost because the device can be in a sleep state at DTIM time. In this mode, there is less power consumption if the beacon listen interval is longer, although the broadcast data can be easily lost.

The *Light-sleep* mode pauses the CPU while keeping the RTC memory, the RTC peripherals and the ultra-low-power (ULP) co-processor are running. The WiFi interface is shut down and the association to the network AP is preserved because the module preserves its internal state. This is not ideal for applications that send WiFi data periodically because the wake time duration, thus data can be lost.

The *Deep-sleep* mode is similar to the Light-sleep mode. The main difference is that the CPU is inactive and WiFi connection data is stored in the RTC memory.

Finally, in the Hibernation mode, all components are shut down, including the internal 8MHz oscillator and ULP processor, except one RTC timer and some RTC GPIOs. Therefore, the device cannot preserve any kind of memory.

Table 3.1 and 3.2 contains the properties of the multiple power modes and the typical current consumption concerning each power mode for ESP32 chip used in this work, respectively.

Table 3.1: ESP32 power modes' hardware level distinction

Item	**Modem-sleep**	**Light-sleep**	**Deep-sleep**	**Hibernation**
WiFi Interface	OFF	OFF	OFF	OFF
AP Association	Connected	Connected	Disconnected	Disconnected
System Clock	ON	OFF	OFF	OFF
RTC	ON	ON	ON	Most OFF
CPU	ON	Pending	OFF	OFF

Table 3.2: Sleep-modes' typical current consumption stated in the datasheet [44]

Power Mode	Description		Power Consumption
Active	Transmit 802.11b		240 μA @ 50% duty
	Transmit 802.11g		190 μA @ 50% duty
	Transmit 802.11n		180 μA @ 50% duty
	Receive 802.11b/g/n		95 mA $\sim$ 100 mA @ 50% duty
Modem-sleep	CPU is powered on	240 MHz	30 mA $\sim$ 68 mA
		160 MHz	27 mA $\sim$ 44 mA
		80 MHz	20 mA $\sim$ 31 mA
Light-sleep	-		0.8 mA
Deep-Sleep	The ULP co-processor is powered on		150 μA
	ULP sensor-monitored pattern		100 μA @ 1% duty
	RTC Timer + RTC memory		10 μA
Hibernation	RTC Timer only		5 μA

3.2.2 MAX30102 Sensor

The MAX30102, the successor of the MAX30100 and MAX30101, is an highly integrated HR monitor and pulse oximeter sensor in a LED reflective solution suitable for wearable devices developed by Maxim Integrated. It has the purpose to acquire HR and S_pO_2.

The sensor contains two LEDs (red and infrared with 660nm and 880nm of wavelength, respectively), photodetectors, cover glass for optimal performance and low-noise electronics for cancelling ambient light. The module operates on a 3.3V power supply. It provides ultra-low-power options with programmable sample rate and LED current as well as a standby mode that has an insignificant current consumption. Communication between the MAX30102 and a microcontroller is via Inter-Integrated Circuit protocol (I^2C). Additionally, the sensor includes a discrete-time filter to reject 50Hz/60Hz noise.

3.2.2.1 Registers

The MAX30102 is fully customised by writing the 8-bits internal registers. Table 3.3 gives an overview of the primary registers.

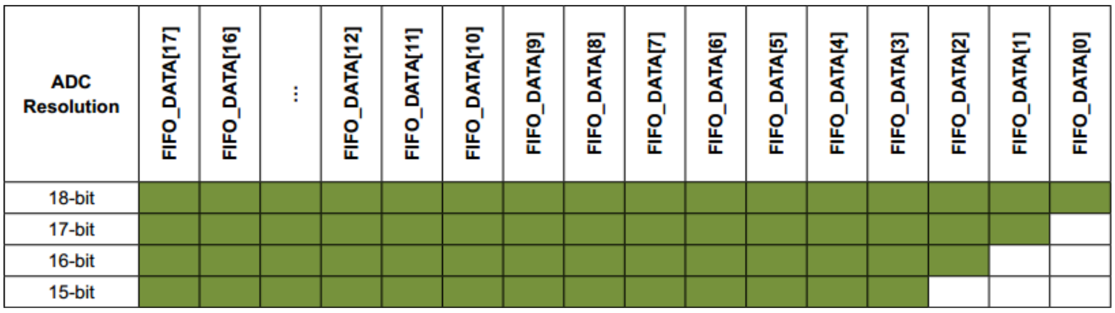
The FIFO is circular and can store up to 32 samples, which is equivalent to 196 bytes (6 bytes per sample). As shown in Figure 3.3, the FIFO data is left-justified, therefore the bit 17 always holds the most significant bit. Figure 3.4 shows the structure of each triplet of bytes (containing the 18-bit ADC data of each LED channel) and visual presentation of how the samples are stored in the FIFO data structure.

3.3 Monitoring Application with GUI

The monitoring application controls the data flow between the main components of the system. Thus, receiving and sending data to the ESP32 through OM2M broker, and updating the InfluxDB with ESP32 data and log activity.

Table 3.3: Overview of the MAX30102's registers

Registers (Address)	Description
FIFO Write Pointer (0x04)	Points to the location where the sensor writes the next sample
Overflow Counter (0x05)	Counts the number of lost samples. When the FIFO is full, samples are not pushed into the FIFO, samples are lost
FIFO Read Pointer (0x06)	Points to the location where the processor gets the next sample from the FIFO through I^2C
FIFO Data Register (0x07)	Stores 8 bits of sample data
FIFO Configuration (0x08)	Set sample averaging and other FIFO behavioural configurations
Mode Configuration (0x09)	Configure operating modes of the sensor: Heart Rate mode (Red LED only), S_pO_2 mode and Multi-LED mode (RED and IR)
S_pO_2 Configuration (0x0A)	Configure S_pO_2 ADC range control, sample rate and LED pulse width
LED Pulse Amplitude (0x0C - 0x0D)	Set current level for Red LED (0x0C) and IR LED (0x0D)
Multi-LED Mode Control (0x11 - 0x12)	In multi-LED mode, each sample is split into four time slots. These registers determine which LED is active in each time slot



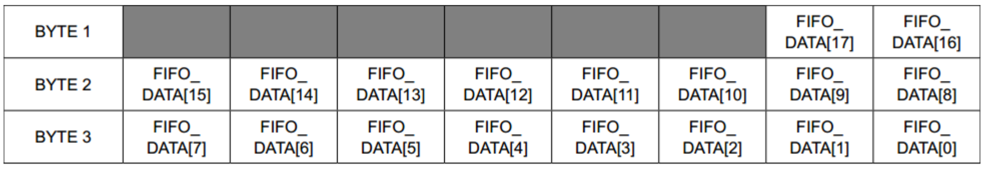
ADC Resolution	FIFO_DATA[17]	FIFO_DATA[16]	⋮	FIFO_DATA[12]	FIFO_DATA[11]	FIFO_DATA[10]	FIFO_DATA[9]	FIFO_DATA[8]	FIFO_DATA[7]	FIFO_DATA[6]	FIFO_DATA[5]	FIFO_DATA[4]	FIFO_DATA[3]	FIFO_DATA[2]	FIFO_DATA[1]	FIFO_DATA[0]
18-bit	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
17-bit	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	
16-bit	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩		
15-bit	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩			

Figure 3.3: Sample data stored in FIFO data structure [70]

Along with the monitoring application, there is a GUI that allows the user to control the system. Both the monitoring application and GUI were developed in Java due to compatibility with the OM2M broker (also a Java implementation).

The GUI interface comprises two pages entitled "Start Page" and "Main Page". The "Start Page", shown in Figure 3.5 is an introductory page with the project's title and a single "Start" button. This button, when clicked, starts the system by initialising communication with the InfluxDB and the OM2M broker, thus enabling the connection with wearables.

The "Main Page" offers real-time information and buttons to control the system. The user can add, pause, remove and delete a single wearable or add and remove all wearables at the same time. The tables "Transmitting Wearables Table" and "Paused Wearables Table" give information on whether a wearable is sending data (active wearables) or not (paused wearable). Finally, the



BYTE 1	▨	▨	▨	▨	▨	▨	FIFO_ DATA[17]	FIFO_ DATA[16]
BYTE 2	FIFO_ DATA[15]	FIFO_ DATA[14]	FIFO_ DATA[13]	FIFO_ DATA[12]	FIFO_ DATA[11]	FIFO_ DATA[10]	FIFO_ DATA[9]	FIFO_ DATA[8]
BYTE 3	FIFO_ DATA[7]	FIFO_ DATA[6]	FIFO_ DATA[5]	FIFO_ DATA[4]	FIFO_ DATA[3]	FIFO_ DATA[2]	FIFO_ DATA[1]	FIFO_ DATA[0]

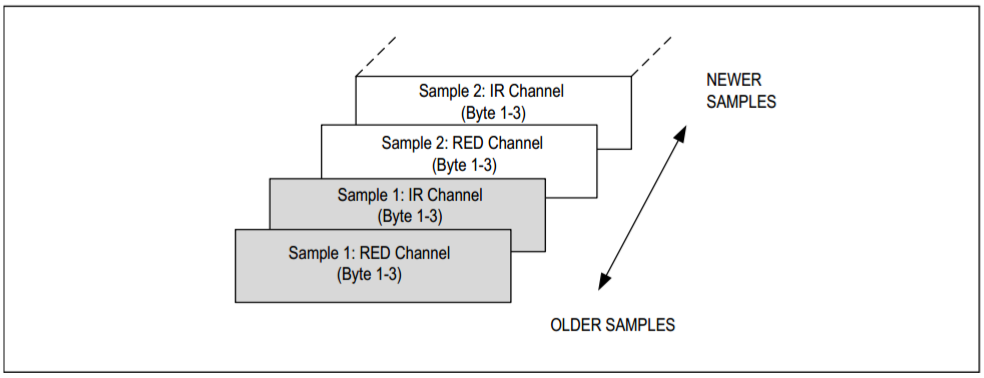


Figure 3.4: FIFO data structure [70]

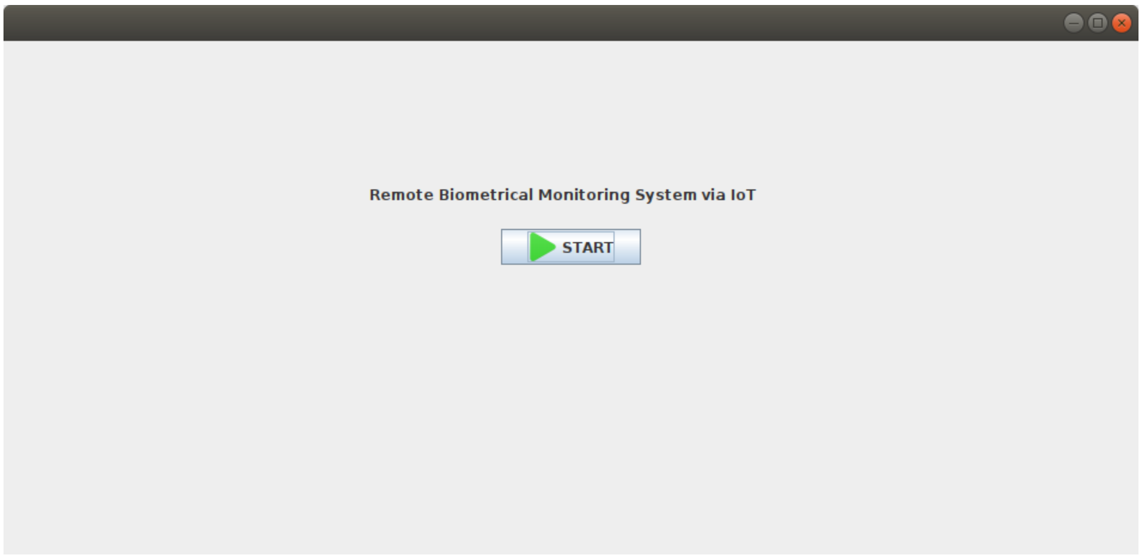


Figure 3.5: GUI’s "Start Page"

button "Chronograf" opens a browser page where the user can visually see all the data regarding a wearable, and the "Restart" restarts the program. Figure 3.6 presents the "Main Page".

3.4 InfluxDB

IoT monitoring systems can generate large amounts of timestamped data in short periods of time, thus using a database specifically built for time-series data is an advantage. From the solu-

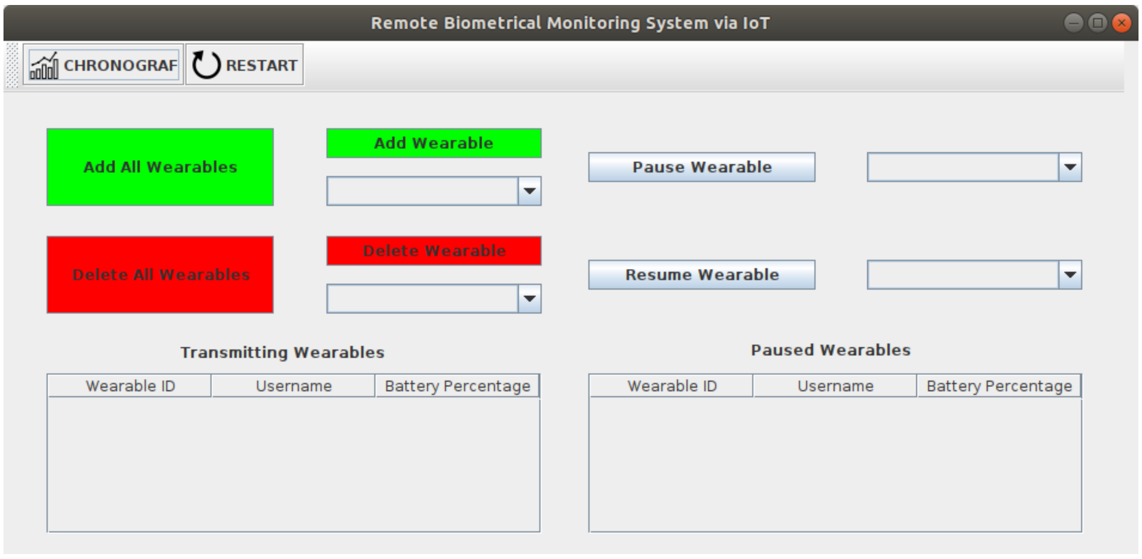


Figure 3.6: GUI's "Main Page"

tions proposed in Section 2.3, InfluxDB [46] is the standout due to its easy to find and available documentation. Therefore, we use this solution to store all the data generated by the system due to its capability to handle timestamped data with high performance, its specific design for an IoT sensor data, write and querying Java HTTP APIs (Java is the programming language for the upper-level end of the system) and SQL-like query language, InfluxQL, developed to query aggregated data. system [47].

All InfluxDB's data have an associated timestamp, therefore there is a column *time* for every InfluxDB instance. This column stores the timestamp aggregated with the data up to nanosecond precision. Data content such as HR values in beats per minute (bpm), is stored in *field values*, which can be strings, floats, integers or booleans. A *field value* is always associated with a timestamp. *Field keys* are strings that *field values* are associated to. For example, an integer *field value* that contains the HR values in bpm is associated with the *field key* "Heart-rate in BPM". In a monitoring system, several users can transmit the same type of data at the same time. *Tag keys* and *tag values* are strings and record metadata. In the example, a *tag key* could be "Username" and, naturally, a *tag values* could be "John Doe". Finally, the fields, tags and *time* compose a *measurement*.

Tags are optional in the data structure, although the use of tags is advisable because tags are indexed, unlike fields. Therefore, query tags are faster, making tags ideal for storing commonly-queried metadata [71].

In this dissertation, we monitor the user's HR and S_pO_2 values and the wearable's battery percentage while differentiating the user. To achieve this requirements, we have an InfluxDB instance named "WearableDB" which has a *measurement* called "measured_data" with three *field keys*: "heart_rate", "oxygen_level", "battery
_percentage", to store for data for HR values in BPM, S_pO_2 values and battery percentage, respectively; and two *tag keys*: "wearable_id" store the wearable identifier (ID) corresponding to the

data transmitted (there is one ID per wearable) and "wearable_username" to store the wearable's username. Table 3.4 presents a visually representation of how the data is stored in the database.

Table 3.4: InfluxDB's instance "WearableDB" for wearable data (illustrative values shown)

measured_data					
time	**heart _rate**	**oxygen _level**	**battery _percentage**	**wearable _id**	**wearable _username**
19:04:32.456	65	95	97	1	John Doe
19:04:32.567	70	96	60	2	Jane Doe
...	...	...	...	...	...

Furthermore, we also store the log information of the monitoring application and GUI. These information is connected to GUI buttons. Therefore, we have an InfluxDB instance named "GuiLogDB" that comprehends a *measurement* called "monitoring_application_gui_log" with one *field key* "operation" to store information related to the pressed buttons, and two *tag keys*: "button" and "page" to indicate the button and the GUI page it belongs, respectively. The "operation" *field key* can inform that the system started when the "Start" button was pressed in the "Start Page" or which wearable (with username and ID) connected to the system, for example. Table 3.5 illustrates the instance to store monitoring and GUI log information.

Table 3.5: InfluxDB's instance "GuiLogDB" for monitoring application and GUI log information

monitoring_application_gui_log			
time	**operation**	**button**	**page**
19:04:33.456	Start System	Start	Start Page
19:04:35.567	Added wearable with username John Doe and ID 1	Add Wearable	Main Page
...	...	...	...

3.5 Chronograf

Chronograph is the visualisation tool for InfluxDB data. Its simple interface, embedded libraries and dashboard templates allow the users to see and analyse data in real-time without needing any knowledge about InfluxDB, which makes it a clear choice for the system.

The dashboards are the centrepieces in Chronograf, which are composed of cells. A cell is a customizable component with different types of graphics, from a line-plot graphics to histograms, allowing the user to choose line colour, legend and notes, among others.

To extract data and visualise from the InfluxDB, the user creates a query. Chronograf has an easy-to-use interface for query creation where the user only has to select an InfluxDB instance, *measurements*, *tags* and *fields* keys for the desired data. We consider this to be a key feature as the user does not need to know the InfluxQL language.

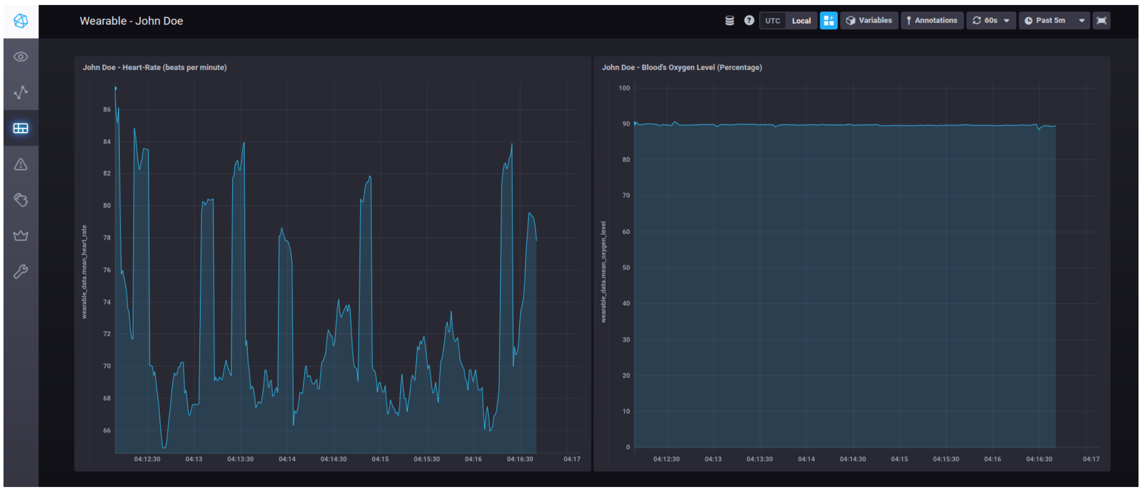

Wearable - John Doe
UTC
Local
Variables
Annotations
60s
Past 5m
John Doe - Heart-Rate (beats per minute)
wearable_data.mean_heart_rate
John Doe - Blood's Oxygen Level (Percentage)
wearable_data.mean_oxygen_level
04:12:30
04:13
04:13:30
04:14
04:14:30
04:15
04:15:30
04:16
04:16:30
04:17

Figure 3.7: Real-time data visualisation of the patient "John Doe"

3.6 Summary

In this chapter, we presented the system's architecture with oneM2M entities and the components that integrate the proposed IoT monitoring system. Moreover, we also presented an overview of each component, namely a description and its function. This system has its central piece in the OM2M broker that ensures device management and data flow between devices. A wearable device, comprising a LiPo battery, an ESP32 and a MAX30102, is used for acquiring the user's physiological parameters and the wearable's battery percentage. On the system's upper-level, we propose a monitoring application with a GUI to give the user the ability to control the system with real-time information display. The wearable data is stored in the time-series database InfluxDB and analysed via Chronograf.

Chapter 4

Implementation and Operation of the IoT Monitoring System

This chapter describes the system's implementation and operation. First, we give an insight on the connection between the MAX30102 and the ESP32 modules in Section 4.1.1. Section 4.1.2 describes the LiPo battery as a power supply. In the Section 4.1.3, we discuss the M2M communications between both the ADN-AE (wearable), the IN-CSE and the IN-AE (monitoring application). We conclude the chapter by describing the system's operation with one and multiple wearables in Section 4.2. For this, we assume a emergency ward scenario where the patient is the user that wears the wearable and the medical personnel is entity that controls the system and analyses data.

4.1 Implementation and Connecting Components

4.1.1 Process Data from the MAX30102 Sensor

The MAX30102 sensor has the purpose of acquiring HR and S_pO_2 data from the user. The data is collected from the sensor and is sent to the ESP32 via I^2C. Figure 4.1 shows the schematic of the connections between the MAX30102 and ESP32.

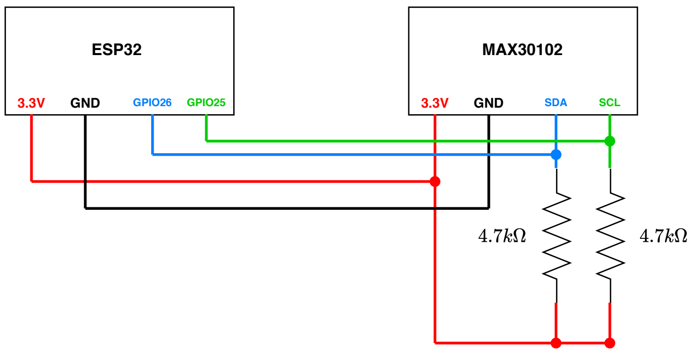


Figure 4.1: Schematic of the connections between the MAX30102 and ESP32

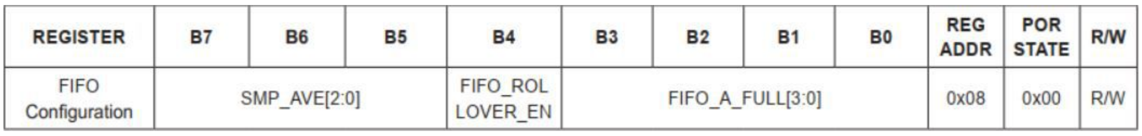
4.1.1.1 Sensor Configuration

The code used for sensor configuration, reading samples and data processing is based on the code developed by the author in [72]. The author proposed a system with an ESP32 and a MAX3010 [73] and conducts an study on power consumption with different sensor configurations with the purpose to minimise power consumption. The author utilises the MAX30100, therefore the code used for this dissertation had to be modified to support the MAX30102 sensor, namely in the register's configuration code. However, both modules are very similar in terms of characteristics, being the Analog-to-Digital Converter (ADC) resolution the only major difference. The MAX30102 has a 18-bit resolution ADC whereas the MAX30100's ADC has a maximum resolution of 15 bits. In this project we use the most-energy efficient configuration, without compromising data accuracy, shown in the Table 4.1.

Table 4.1: MAX30102 configuration parameters

Parameter	Value
Nº of samples average per FIFO sample*	32
Mode Configuration	HR and S_pO_2
S_pO_2 ADC range control*	2048 nA
S_pO_2 sample rate control	50 samples per second
LED pulse width control (ADC resolution)*	15 bits
Red LED current	27.2 mA
IR LED current	43.6 mA

The parameters marked with the "*" symbol represent parameters adapted for the MAX30102 or do not exist in the MAX30100. The number of samples average per FIFO sample is set in the three most significant bits (MSB) of the FIFO configuration register, shown in Figure 4.2, which does not exist in the MAX30100. Table 4.2 presents the minimum and maximum values for number of samples averaged per FIFO sample. For this project, the bits are set to 32 to reduce the throughput data. For a description of the MAX30102's registers refer to Section 3.2.2.1.



REGISTER	B7	B6	B5	B4	B3	B2	B1	B0	REG ADDR	POR STATE	R/W
FIFO Configuration	SMP_AVE[2:0]			FIFO_ROLLOVER_EN	FIFO_A_FULL[3:0]				0x08	0x00	R/W

Figure 4.2: MAX30102 FIFO Configuration register [70]

The ADC is set to the lowest resolution possible, i.e. 15 bits, because the most-energy efficient configuration for the MAX30100 has a 13-bit ADC resolution, which is the lowest resolution for the module. For the same reason, S_pO_2 ADC range control has the lowest value, although this parameter does not exist in the MAX30100.

Additionally, the MAX30102 allows the developer to configure the FIFO behaviour when completely full whereas this feature is not configurable on the MAX30100. The bit FIFO_ROLLOVER_EN, in the FIFO Configuration register, controls whether the FIFO holds the data until FIFO Data is

Table 4.2: Number of samples averaged per FIFO sample in the FIFO Configuration register

SMP_AVG[2:0]	**Nº of samples averaged per FIFO sample**
000	1 (no averaging)
001	2
010	4
011	8
100	16
101	32
110	32
111	32

read or the FIFO Write/Read pointer positions are changed by setting the bit to 0 (zero), or stores new data by changing the FIFO address to zero by setting the bit to 1 (one). For this project, it is important to analyse the newest data from the patients, thus setting the bit is set to 1 (one).

4.1.1.2 Sensor Data Handling in the ESP Code

There is only one function in the ESP's programming to handle the secondary functions to acquire sensor data. This task configures the ESP32 I^2C driver and communication with sensor, and reads and processes the data from the sensor. Moreover, after a new HR and S_pO_2 measurement, the task publishes the new data to the IN-CSE. Lastly, this task has a period of one second and has the highest priority when compared to other system's tasks.

4.1.2 LiPo Battery Power Supply

For powering the wearable we use a single-cell LiPo battery with 3.7V, with 4.2V of peak voltage. Figure 4.3 demonstrates the wearable's power schematic. Both the ESP32 module and MAX30102 operate on 3.3V, hence an MCP1702-3302ET Low-dropout regulator (LDO) with a fixed output voltage of 3.3V. This LDO has 1.6 μA typical quiescent current and 250 mA output current. The electrolytic and ceramic capacitors smooth the voltage peaks.

4.1.2.1 Battery Percentage

When using battery-powered devices, it is useful to monitor the battery level. Reading the output voltage of the battery from an analog pin of the ESP32. However, the ESP32 GPIO's work at 3.3V, thus the need for a voltage divider, as shown in Figure 4.4.

LiPo batteries have a well-known State of Charge (SoC), which is linear function with ADC read voltage as a function of battery percentage. The LiPo battery is completely charged at 4.2V, therefore one-hundred (100) percent, and completely empty, zero (0) percent, at 3.5V.

To convert the voltage into a percentage value, we perform two linear conversions.

First, we scale the values read by the ADC analog pin to the real voltage values at the battery terminals. The values read in the ADC are 12 (twelve) bits wide, therefore with a 0 to 4095

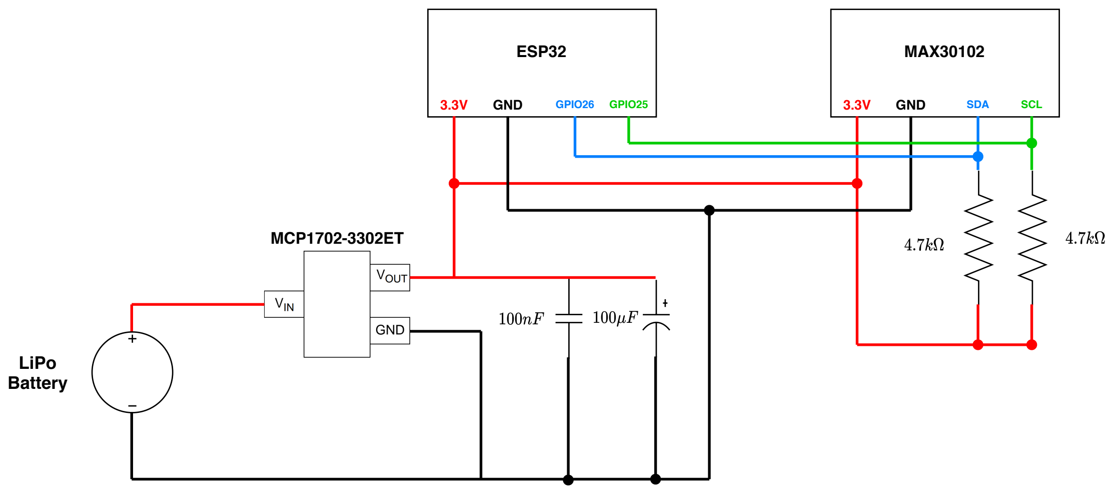


Figure 4.3: Schematic of the connections between the wearable's components

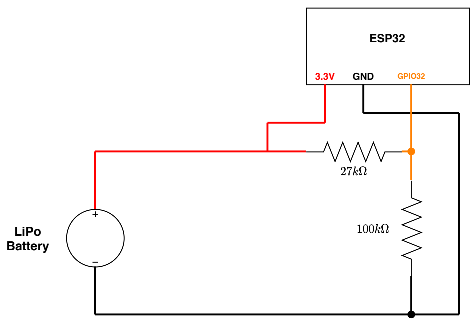


Figure 4.4: Schematic of the connections with the voltage divider to acquire battery voltage through the analog pin

range values, which represent the "adc_min_value" and "adc_max_value", respectively. The variable "adc_value_read" represents the value read from the analog pin. The real battery voltage value is between 3.5V and 4.2V, thus 3.5V equals "battery_min_value" and 4.2V equals "battery_max_value". The variable "battery_value" is the actual voltage value at the battery's terminals. Equation 4.3 presents the formula to transform the voltage value read by the ADC in the actual voltage at the battery terminals.

$$adc\_range = adc\_min\_value - adc\_max\_value \tag{4.1}$$

$$battery\_range = battery\_min\_value - battery\_max\_value \tag{4.2}$$

$$\begin{aligned} battery\_value &= \frac{(adc\_value\_read - adc\_min\_value) * battery\_range}{adc\_range} + battery\_min\_value \\ &= \frac{(adc\_value\_read - 0) * (4.2 - 3.5)}{4095 - 0} + 3.5 \end{aligned} \tag{4.3}$$

The second and final linear conversion, transforms the "battery_value" in battery percentage with the Equation 4.5.

$$percent\_range = percent\_min\_value - percent\_max\_value \tag{4.4}$$

$$\begin{aligned} battery\_percent &= \frac{(battery\_value - battery\_min\_value) * percent\_range}{battery\_range} + percent\_min\_value \\ &= \frac{(battery\_value - 3.5) * (100 - 0)}{4.2 - 3.5} + 0 \\ &= \frac{(battery\_value - 3.5) * (100)}{0.7} \end{aligned} \tag{4.5}$$

Regarding code implementation, the program defines a periodic task that reads the output voltage from the ADC pin every second. After converting the ADC value to percentage, this new battery percentage value is compared to the previous one acquired, thus publishing battery percentage value only if the values are different, reducing network traffic and power consumption.

4.1.3 M2M Communications Between the AEs and the IN-CSE

4.1.3.1 ADN-AE and IN-CSE

The IN-CSE supports CoAP, HTTP and MQTT protocols. In this project, the ADN-AE, i.e. the wearable's software, publishes the data collected at the IN-CSE using the MQTT protocol. We use MQTT due to the advantage of its communication model, publisher-subscriber, in terms of communication efficiency in sensing and remote monitoring applications. Although, the IN-CSE supports MQTT communications, an MQTT server is needed to act as a broker for communications between the two entities. The system utilises the Eclipse Mosquitto MQTT server. This server was installed and configured in the machine where the IN-CSE is running.

An oneM2M standard request requires obligatory parameters that need to be included in the MQTT message payload (Figure 4.5), namely:

- **To** (to): Contains receiver's URI;
- **From** (fr): Contains sender's ID;
- **Operation** (op): Indicates operation's request type. There are six different types of operation: Create(1), Retrieve(2), Update(3), Delete(4), Notify(5) and Discovery(6);

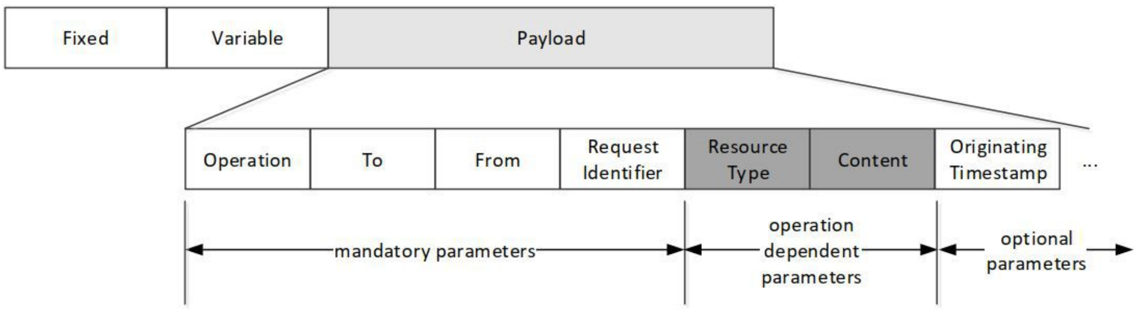


Figure 4.5: MQTT's message payload with oneM2M parameters [74]

Besides the previous parameters, there are other additional parameters depending on the type of operation. In order to publish the data to the IN-CSE, we use generic functions, developed by the author in [75], to create the MQTT's message payload depending on the type of operation. This functions allow the register AEs, containers where the information is stored, content instances in a existing containers and, finally, subscriber entities in the IN-CSE.

Additionally, the software implemented in the ADN-AE uses the Eclipse Paho library [76] for MQTT communication with the broker.

4.1.3.2 IN-AE and IN-CSE

The communication between the IN-AE, i.e. the monitoring application, and the IN-CSE goes through HTTP protocol. The software implementation for create AE, containers, content instances and subscriber entities as well as receiving data from the IN-CSE was based and adapted from software presented on the Eclipse OM2M official website [77].

4.1.3.3 IN-CSE Data Structure

For the ADN-AE, an AE is registered in the IN-CSE with name "Wearable_IP", where the "IP" represents the device's IP in the network, thus never having wearable device's registered with the same AE name, which allows multiple wearable connected to the system and active at the same time. Associated to "Wearable_IP" are the following containers, mainly to store acquired data and published by the wearable to the IN-CSE:

- **Heart_Rate**: Stores every content instance associated to the patient's HR;
- **Oxygen_Level**: Stores every content instance associated to the patient's S_pO_2;

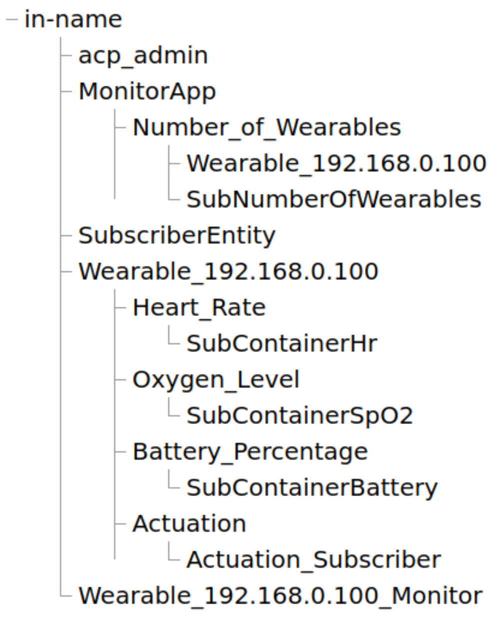
- **Battery_Percentage**: Stores every content instance associated with the wearable's battery percentage;

- **Actuation**: Receives "START" and "STOP" signals for start and stop publishing data to the IN-CSE from the monitoring application;

Additionally, the ADN-AE registers a subscriber entity "Wearable_IP_Monitor" to subscribe that from the "Actuation" container. For this, an entity "Actuation_Subscriber", associated with the "Wearable_IP_Monitor", is created in the "Actuation" container to subscribe the data published in that container.

Lastly, the IN-AE registers an AE in the IN-CSE with the name "MonitorApp" with a associated container, "Number_of_Wearable", to store information about the wearables connected to the system, although this information is sent by the wearable, which means that the monitoring application subscribes to the container by creating a subscriber entity name "SubscriberEntity" with an entity "SubNumberOfWearables" associated to the container. Moreover, the monitoring application has to subscribe to the wearable's data containers to receive data, therefore creating content instances "SubContainerBattery", "SubContainerHR" and "SubContainerOxygenLevel", associated to "SubscriberEntity".

Figure 4.6 shows the data registered in the IN-CSE for the one wearable connected to the system.



```
- in-name
  ├ acp_admin
  ├ MonitorApp
  │  ├ Number_of_Wearables
  │  │  ├ Wearable_192.168.0.100
  │  │  └ SubNumberOfWearables
  ├ SubscriberEntity
  ├ Wearable_192.168.0.100
  │  ├ Heart_Rate
  │  │  └ SubContainerHr
  │  ├ Oxygen_Level
  │  │  └ SubContainerSpO2
  │  ├ Battery_Percentage
  │  │  └ SubContainerBattery
  │  ├ Actuation
  │  │  └ Actuation_Subscriber
  └ Wearable_192.168.0.100_Monitor
```

Figure 4.6: IN-CSE data structure

4.2 System Operation

4.2.1 Setup Stage

In this section, we describe the system operating with a single wearable connecting and transmitting data to the monitoring application, although the system can handle multiple wearable devices at the same time.

First, the medical personnel has to start the monitoring application and, consequently, the GUI before turning on any wearable. This is a system requirement. After initialising the application, the "Start Page" of the GUI (Figure 3.5) appears. Now, the medical personnel has to press the "START" button to allow the wearable to connect to the system. The pressing of the button triggers multiple actions, as illustrated in Figure 4.7:

- Registers, in the IN-CSE, the entities for the monitoring application discussed in Section 4.1.3.3;
- Initiates communication, through HTTP, with InfluxDB;
- Registers in the database instance "GuiLogDB" the start of the system;
- Opens the GUI's "Main Page".

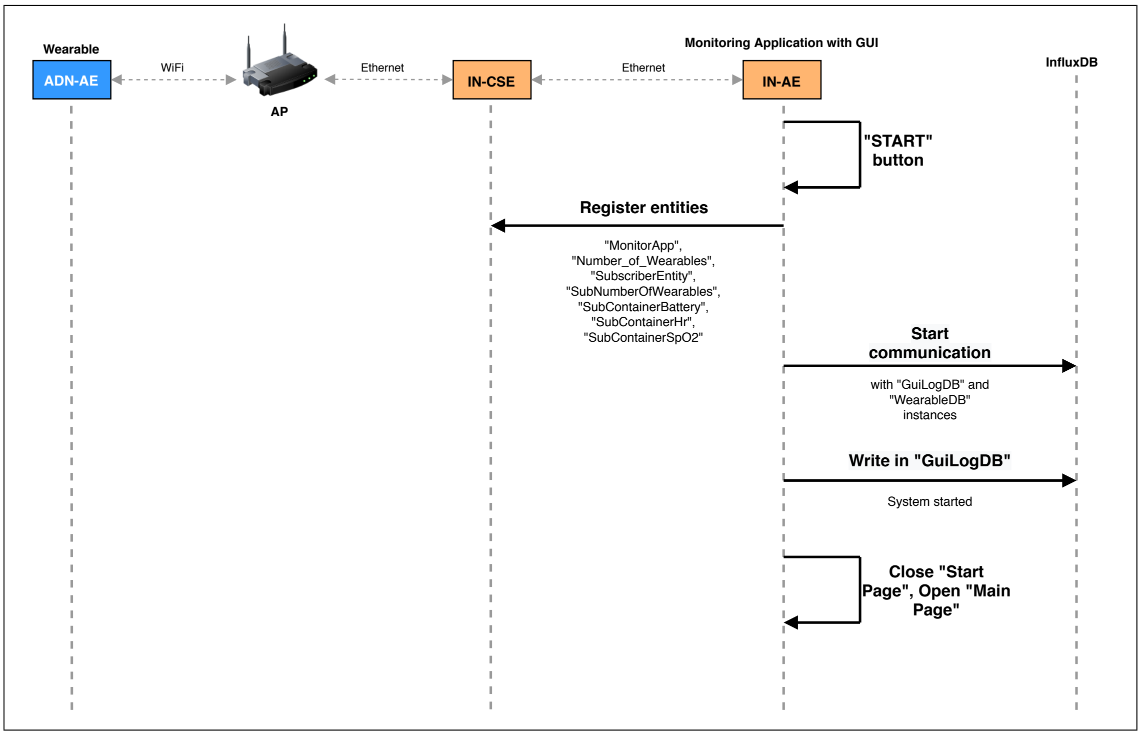

Wearable
ADN-AE
WiFi
AP
Ethernet
IN-CSE
Ethernet
Monitoring Application with GUI
IN-AE
InfluxDB
"START" button
Register entities
"MonitorApp", "Number_of_Wearables", "SubscriberEntity", "SubNumberOfWearables", "SubContainerBattery", "SubContainerHr", "SubContainerSpO2"
Start communication
with "GuiLogDB" and "WearableDB" instances
Write in "GuiLogDB"
System started
Close "Start Page", Open "Main Page"

Figure 4.7: Sequence diagram for the actions performed by the monitoring application after pressing the "START" button

Only now the wearable can be initiated. After its initialisation, it creates the corresponding entities referred in the Section 4.1.3.3. The wearable creates a content instance in the "Number_of_Wearables" with the wearable's username and IP, and then waits for a "START" signal from the monitoring application to start transmitting, which is a content instance in the "Actuation" container. After the monitoring application receives the content instance published in the "Number_of_Wearables", the application creates a wearable object with the wearable's username and IP as well as an ID and other status variables. Although the wearable IP is already an unique identifier, most medical personnel don't have knowledge about IP addresses, thus creating an ID number as it is more visually appealing. In the combinational box below the "Add Wearable" button, a new option appears with the wearable's ID and username, as shown in Figure 4.8.

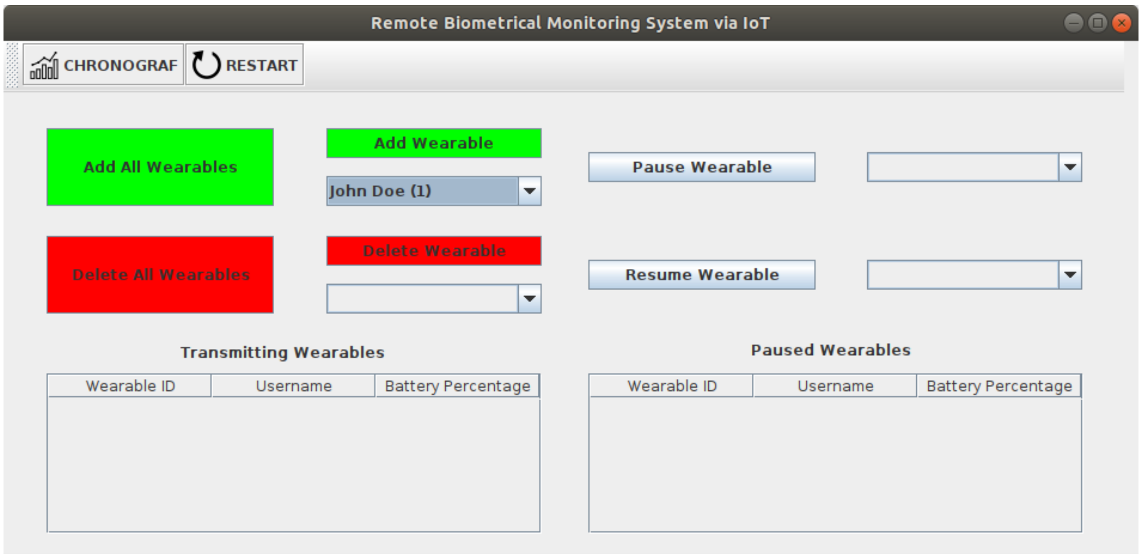
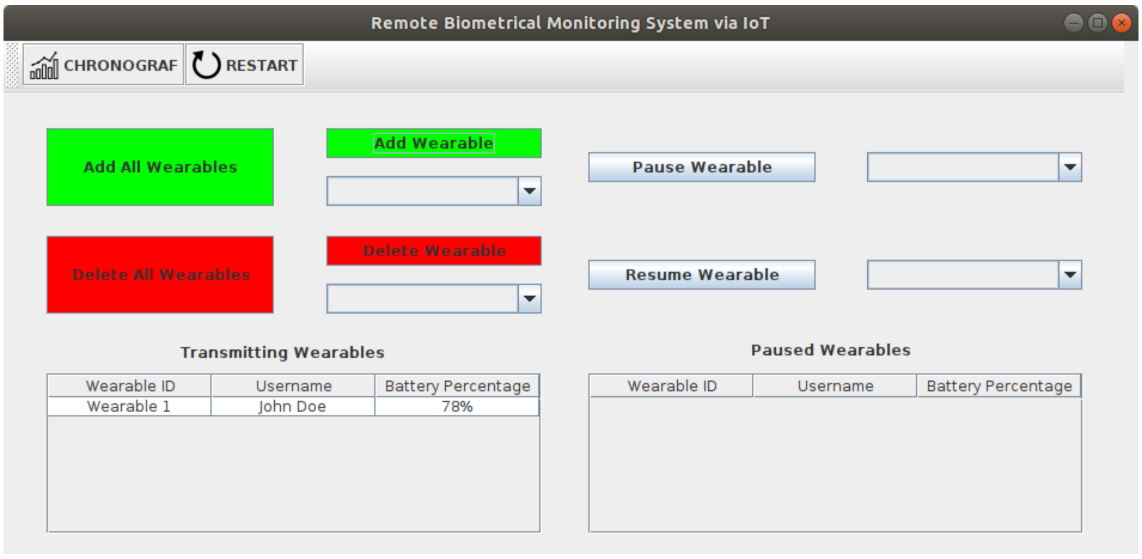


Figure 4.8: GUI's "Main Page" with the wearable waiting to be added to the system

At this moment, the medical personnel can decide to add the wearable to the monitoring the system or not. If they decide not to add the wearable, they have the option to add it at a later time through the interface. On the other hand, to add the wearable to the system, the medical personnel has to select the wearable in the combinational box and press the "Add Wearable" button. This action sends a "START" signal to the wearable, which enables it to start publishing data, and adds a line in the table "Transmitting Wearables" with the wearable's ID, username and battery percentage (Figure 4.9). This last value is updated whenever the wearable publishes a new content instance with a new battery percentage value. Additionally, a new data point with the wearable's information, namely the ID, IP and username, and the button pressed is inserted in the database instance "GUILogDB".

The wearable is now publishing data to the respective containers and, consequently, the monitoring application is receiving the data. After receiving a content instance of any data, the monitoring application inserts a data point in the database instance "WearablesDB" with the new data value. Figure 4.10 illustrates the actions performed between the moments where the wearable is initialised and transmits data.



Transmitting Wearables

Wearable ID	Username	Battery Percentage
Wearable 1	John Doe	78%

Paused Wearables

Wearable ID	Username	Battery Percentage

Figure 4.9: GUI's "Main Page" with the wearable transmitting data

4.2.2 Device & Data Management During Operation

The medical personnel has two options regarding the wearable status: pause/resume the wearable or delete it from the system. First, we focus on the pause/resume wearable.

After pressing the "Add Wearable" button, a new option appears in the combinational box on the right of the "Pause Wearable" button. If the medical personnel selects the wearable and presses button, the monitoring application sends a "STOP" signal, i.e, a content instance in the "Actuation" container, to inform the wearable to stop publishing data. Now the wearable is paused, therefore the monitoring application inserts a new line in the "Paused Wearables" table with the wearable's information and deletes the corresponding line in the "Transmitting Wearables" table (Figure 4.11). Similarly to the previous buttons, a new data point is stored in the "GUILogDB". When the wearable is paused, the medical personnel can resume its previous state. To do this, the medical personnel has to select the wearable in the combinational box at the right of the "Resume Button" and, when selected, press the button. The monitoring application sends a new "START" signal to the wearable, deletes the respective line from the table "Paused Wearables" and adds that line to the "Transmitting Wearables" table, as shown in Figure 4.9, and writes a new data point in the "GuiLogDB" instance that the respective wearable has resumed activity.

Figure 4.12 presents the actions performed for pausing and resuming wearable activity.

After adding the wearable to the system, the medical personnel can delete the wearable from the system, even if the wearable is paused. Analogous the other actions, the medical personnel has to select the desired wearable in the combinational box below the "Delete Wearable" button and press the button. After the button is pressed, the monitoring application sends a new "STOP" signal and deletes all the entities in the IN-CSE related to the wearable as well as deletes the line from one of the GUI tables, depending the wearable's status. Figure 4.13 shows the actions after pressing "Delete Wearable".

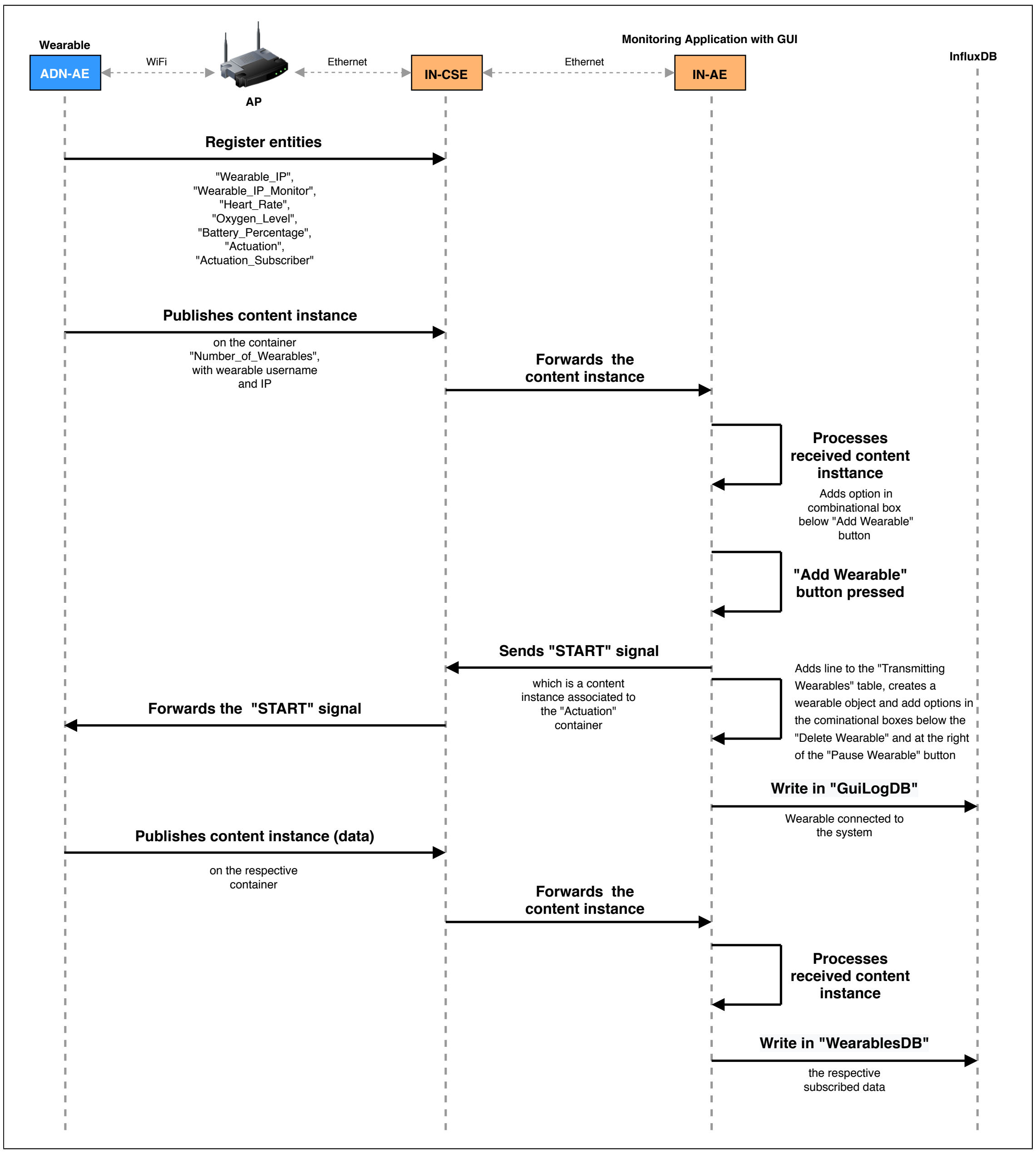
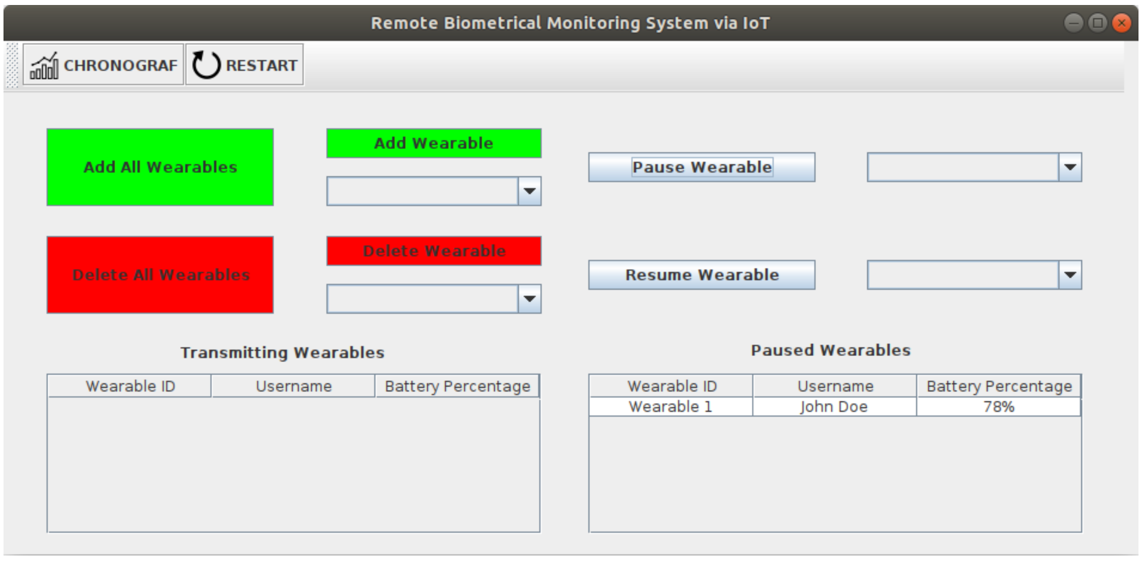

Wearable
ADN-AE
WiFi
AP
Ethernet
IN-CSE
Monitoring Application with GUI
Ethernet
IN-AE
InfluxDB
Register entities
"Wearable_IP", "Wearable_IP_Monitor", "Heart_Rate", "Oxygen_Level", "Battery_Percentage", "Actuation", "Actuation_Subscriber"
Publishes content instance
on the container "Number_of_Wearables", with wearable username and IP
Forwards the content instance
Processes received content insttance
Adds option in combinational box below "Add Wearable" button
"Add Wearable" button pressed
Sends "START" signal
which is a content instance associated to the "Actuation" container
Forwards the "START" signal
Adds line to the "Transmitting Wearables" table, creates a wearable object and add options in the cominational boxes below the "Delete Wearable" and at the right of the "Pause Wearable" button
Write in "GuiLogDB"
Wearable connected to the system
Publishes content instance (data)
on the respective container
Forwards the content instance
Processes received content instance
Write in "WearablesDB"
the respective subscribed data

Figure 4.10: Sequence diagram to add a wearable to the system

Lastly, the medical personnel can restart the system by clicking the "RESTART" button at any time. The monitoring application deletes the all the entities registered in the IN-CSE and creates again the entities referred to the monitoring application discussed in the Section 4.1.3.3

4.2.3 Data Visualisation

The data published from the wearable and subscribed by the monitoring application are stored in the database instance "WearablesDB" associated with the wearable's username and ID. The data stored can be visualised by the medical personnel by pressing the button "Chronograf".



Transmitting Wearables

Wearable ID	Username	Battery Percentage

Paused Wearables

Wearable ID	Username	Battery Percentage
Wearable 1	John Doe	78%

Figure 4.11: GUI's "Main Page" with the paused wearable

After pressing the button, a browser page opens up with a page to create a dashboard for a user. The medical personnel needs to clone the template dashboard, with the name "Wearable - Patient", for each patient. By clicking in the created dashboard, the medical personnel is redirected to a new page with two empty cells with the name "Patient - Heart-Rate (beats per minute)" and "Patient - Blood's Oxygen Level (Percentage)", as shown in Figure 4.14. In this page, the medical personnel can choose the data refresh rate among other functionalities.

A type of data is associated to each cell. Therefore, by editing the cell, the medical personnel can now associate the data from the database and create graphics. It is the cell configure page that a graphic with a type of data is created. For example, in Figure 4.15 the cell presents the HR data from the wearable with the username "John Doe". In the same page, the data graphic is fully customizable on the banner "Visualization" (Figure 4.16).

Figure 4.17 shows the dashboard real-time data of the patient "John Doe".

4.2.4 System with multiple wearables

As stressed earlier, the system proposed can have multiple wearables connected and transmitting at the same time. Therefore, the system has functionality to add all the wearables at the same time, which is embedded in the "Add All Wearables" button in the "Main Page". The process to add all the wearables to the system is the same presented in Figure 4.10 repeated the number of active wearables. However, the medical personnel can add a wearable individually.

Additionally, the medical personnel can delete all system's wearables at the same time by clicking the "Delete All Wearables" in the "Main Page". Analogous to adding all wearables at the same time, the process to delete all wearables is the process presented in Figure 4.13 repeated the number of active wearables.

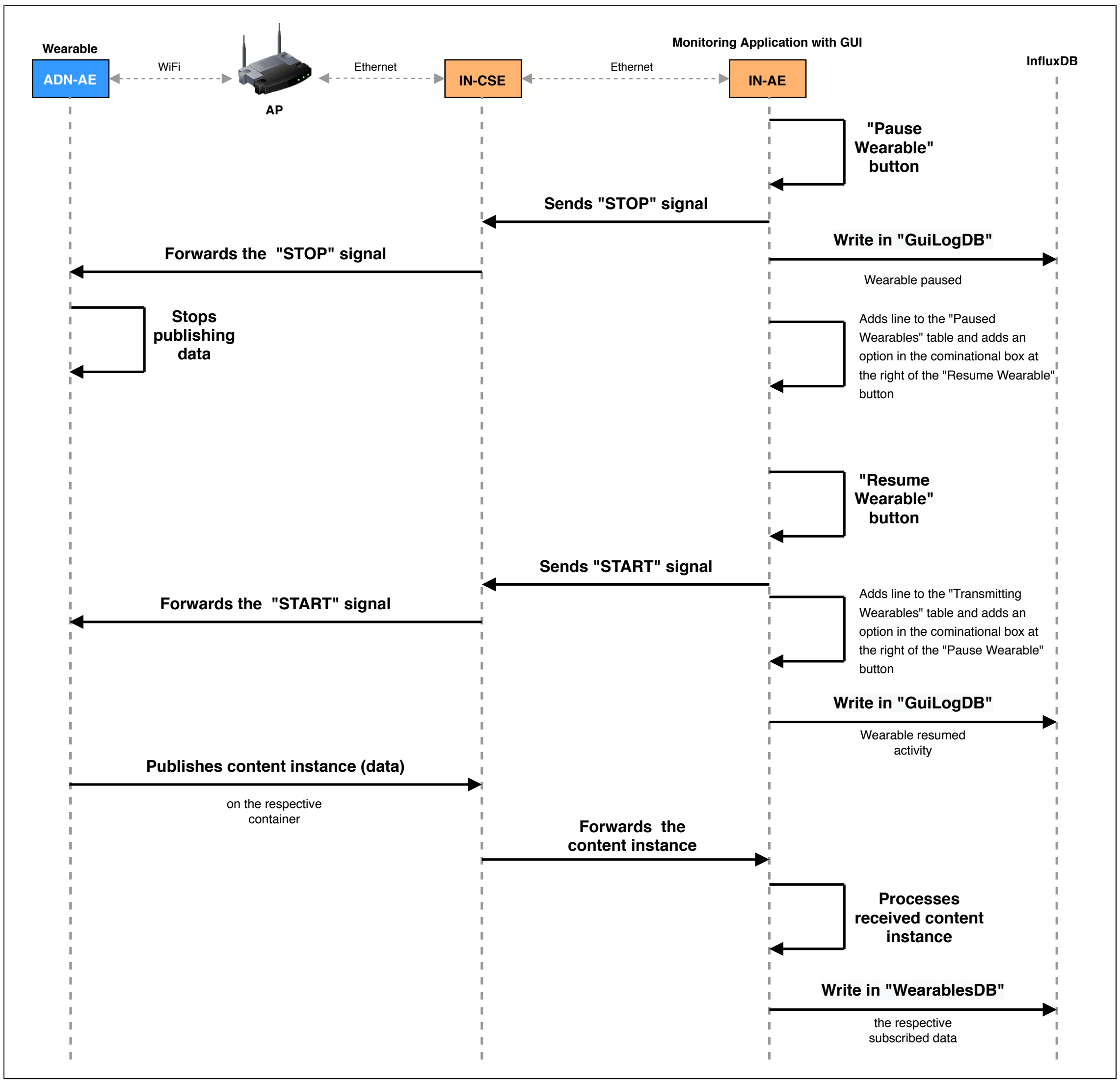

Wearable
ADN-AE
WiFi
AP
Ethernet
IN-CSE
Ethernet
Monitoring Application with GUI
IN-AE
InfluxDB
"Pause Wearable" button
Sends "STOP" signal
Write in "GuiLogDB"
Wearable paused
Forwards the "STOP" signal
Stops publishing data
Adds line to the "Paused Wearables" table and adds an option in the cominational box at the right of the "Resume Wearable" button
"Resume Wearable" button
Sends "START" signal
Forwards the "START" signal
Adds line to the "Transmitting Wearables" table and adds an option in the cominational box at the right of the "Pause Wearable" button
Write in "GuiLogDB"
Wearable resumed activity
Publishes content instance (data)
on the respective container
Forwards the content instance
Processes received content instance
Write in "WearablesDB"
the respective subscribed data

Figure 4.12: Sequence diagram to pause and resume a wearable

4.3 Summary

In this chapter, we presented the most efficient configuration for the MAX30102, energy wise, based on the study in [72] as well as the adaptations needed for this specific. We also discussed the the power schematic for a LiPo battery powered device as well as the schematic and formulas to calculate battery percentage. The architecture includes a set of entities and containers registered to the IN-CSE needed for the system operation. Lastly, we discussed the application of our system to an emergency ward scenario with only one wearable first and then with multiple wearables.

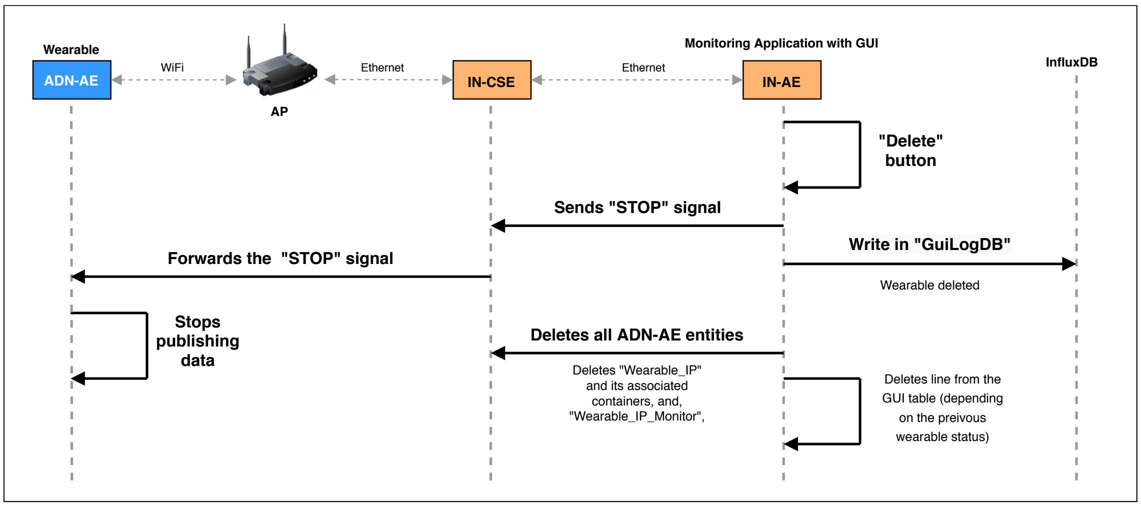

Wearable
ADN-AE
WiFi
AP
Ethernet
IN-CSE
Ethernet
Monitoring Application with GUI
IN-AE
InfluxDB
"Delete" button
Sends "STOP" signal
Forwards the "STOP" signal
Write in "GuiLogDB"
Wearable deleted
Stops publishing data
Deletes all ADN-AE entities
Deletes "Wearable_IP" and its associated containers, and, "Wearable_IP_Monitor",
Deletes line from the GUI table (depending on the preivous wearable status)

Figure 4.13: Sequence diagram to delete a wearable from the system

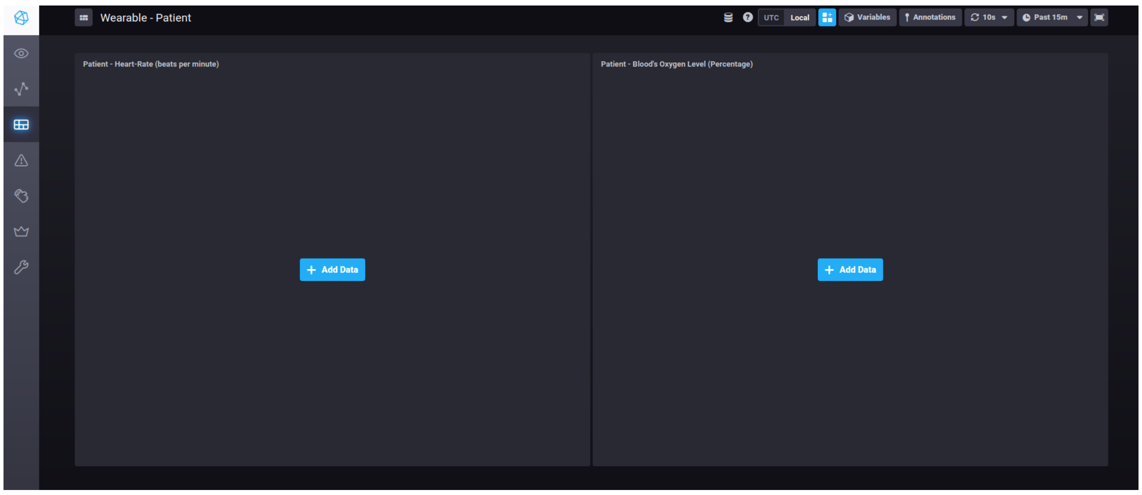
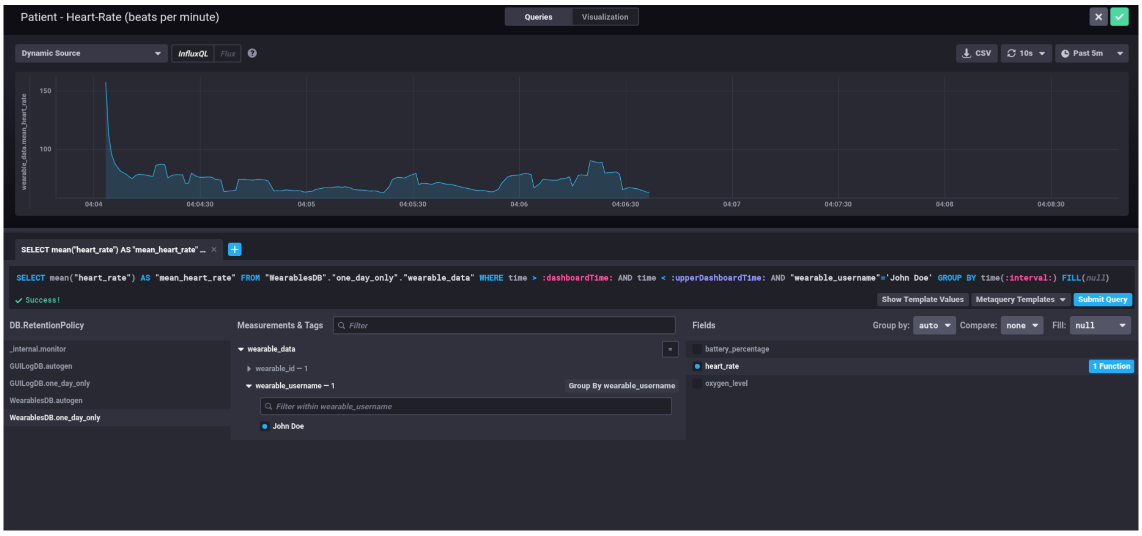

Wearable - Patient
UTC
Local
Variables
Annotations
10s
Past 15m
Patient - Heart-Rate (beats per minute)
Patient - Blood's Oxygen Level (Percentage)
Add Data
Add Data

Figure 4.14: Chronograf's patient dashboard



```
SELECT mean("heart_rate") AS "mean_heart_rate" FROM "WearablesDB"."one_day_only"."wearable_data" WHERE time > :dashboardTime: AND time < :upperDashboardTime: AND "wearable_username"='John Doe' GROUP BY time(:interval:) FILL(null)
✓ Success!
```

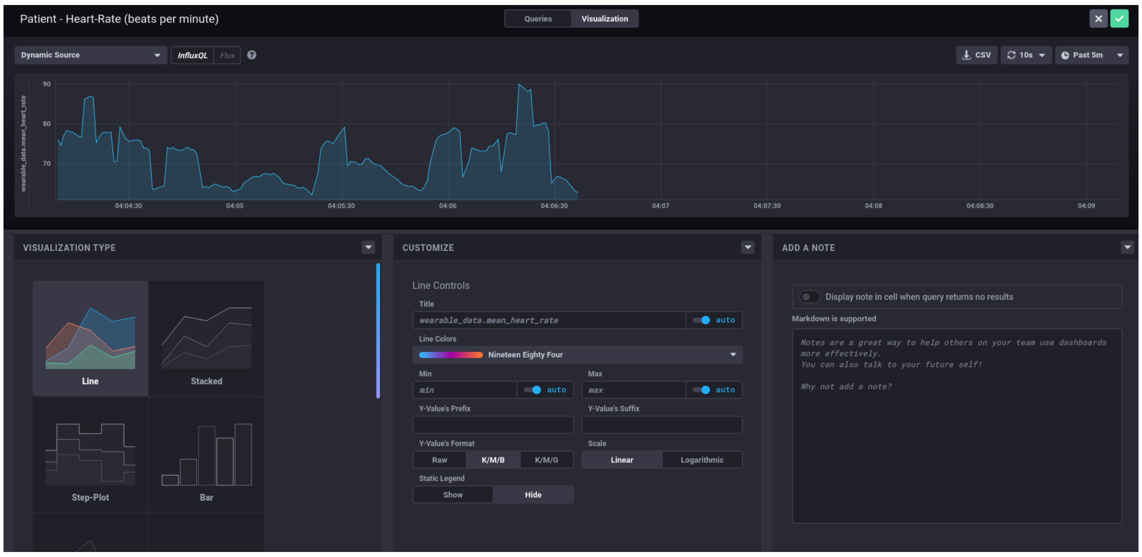
DB.RetentionPolicy: _internal.monitor, GUILogDB.autogen, GUILogDB.one_day_only, WearablesDB.autogen, WearablesDB.one_day_only

Measurements & Tags: wearable_data; wearable_id — 1; wearable_username — 1 (Group By wearable_username); John Doe

Fields: battery_percentage, heart_rate, oxygen_level

Group by: auto Compare: none Fill: null

Figure 4.15: Dashboard cell's area to submit a query to InfluxDB



Line Controls

Title: wearable_data.mean_heart_rate (auto)

Line Colors: Nineteen Eighty Four

Min: min (auto) Max: max (auto)

Y-Value's Prefix:

Y-Value's Suffix:

Y-Value's Format: Raw [✓ K/M/B] K/M/G

Scale: [✓ Linear] Logarithmic

Static Legend: Show [✓ Hide]

Display note in cell when query returns no results

Markdown is supported

Notes are a great way to help others on your team use dashboards more effectively.
You can also talk to your future self!

Why not add a note?

Figure 4.16: Dashboard cell's visualization area

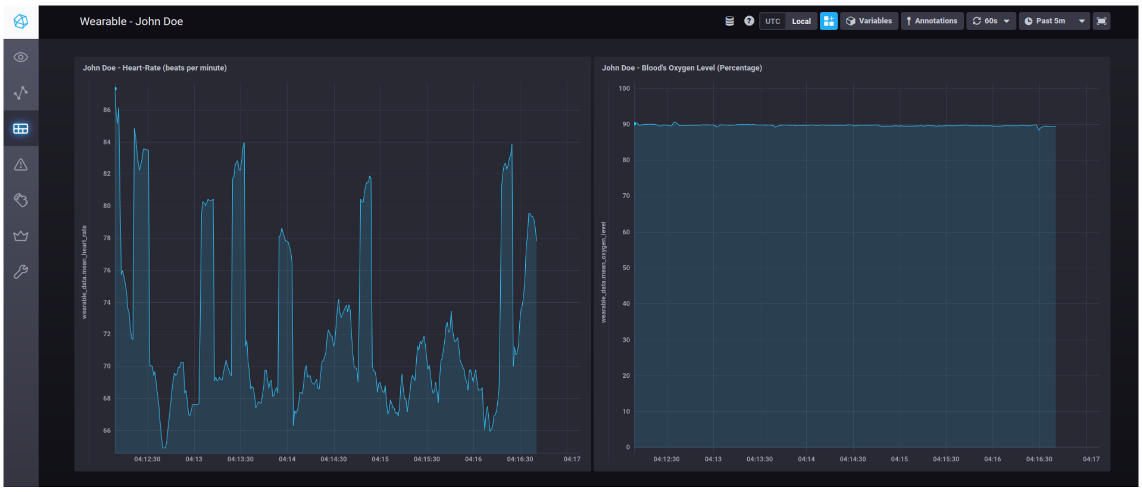


Figure 4.17: Real-time data visualization of the patient "John Doe"

Chapter 5

Performance Results

The use of M2M middlewares as the oneM2M are major steps forward in the development of in developing M2M and IoT application and promoting interoperability and standardisation. However, this brings additional overhead and delay on the communications between the devices. The additional amount of information that has to be added in each transmission and, consequently, for the additional time that each transmission needs to be completed reflects the problem at hand. Therefore, in the chapter we evaluate the ESP32 in this scenario with E2E latency and PDR for each protocol (CoAP, HTTP and MQTT) and ESP32's power mode.

5.1 Setup and Methodology

We conducted the performance experiments for the ESP32 module in an indoor environment. The IN-CSE and the MQTT server were installed in a dedicated machine with Ubuntu 18.04LTS OS and an Intel® Core™i5-3337U CPU @ 1.80GHz quadcore processor and 4GB of RAM. The IN-AE (subscriber), installed in a computer with the same OS as the server but with an Intel® Core™i7-7700HQ CPU @ 2.80GHz octacore processor and 16GB of RAM, was connected to the server machine with a 100Mb/s Ethernet connection. The ADN-AE (ESP32 module) was connected an infrastructured local WiFi network with a TP-Link TL-WR940N multi-mode router as an AP configured to transmit a Beacon every 100ms and a DTIM message every 3 Beacons. Lastly, the AP was connected to the IN-CSE machine with a 100Mb/s Ethernet connection.

The ADN-AE and the IN-AE are not installed in the same machine, therefore we used a Network Time Protocol (NTP) server, installed on the IN-CSE machine, to synchronise the AEs clocks. Both applications re-synchronise with the server every 10 minutes. This re-sycnhronize time interval value is the minimum value allowed by the ESP-IDF framework.

We conducted the experiments regarding the WiFi communication channel quality. At first, the ADN-AE publishes data very close to the AP with a Received Signal Strength Indication (RSSI) value of -30, which we denominate as *good* signal strength channel experiment. For the *bad* signal strength channel experiment, the ADN-AE published the data further away from the AP with a RSSI value of -85. For each channel quality experiment, the ADN-AE publishes data with different payload and different frequency. We assume that one HR sample has 85B of payload, therefore we study the scenarios where the ADN-AE publishes 85B of payload samples every second and 850B of payload (corresponding to 10 samples) every 10 seconds. Moreover, we evaluate the impact of three protocols, CoAP, HTTP and MQTT, for both channel quality experiments. Moreover, for the CoAP experiments, we utilised its non-confirmable version (CoAP non-confirmable), and for MQTT experiments we evaluate its three QoS of service oriented variants: MQTT QoS0, MQTT QoS1 and MQTT QoS2. Also, we measure the impact of the ESP32 module's available power modes, *Active-sleep* mode, *Minimum Modem sleep* mode and *Max Modem sleep* mode. For the Modem mode variants, the ESP-IDF framework offers power management feature that allows the ESP32 module to automatically go to *Light-sleep* mode. For each scenario, we conducted experiments until the ADN-AE publishes 100 messages.

Finally, the timestamps are registered before the ADN-AE publishes the message and after the IN-AE receives and processes the message.

5.2 End-to-end Latency Experiments

In Sections 5.2.1, 5.2.2 and 5.2.3 we present E2E latency and PDR results on the considered scenarios for CoAP non-confirmable, HTTP and the three MQTT variants, respectively. In the Section 5.2.4, we discuss and compare the results.

Finally, the graphic legends to distinct the power modes have the following meaning:

- **Active**: *Active-sleep* mode;
- **MinModem**: *Minimum Modem sleep* mode;
- **MaxModem**: *Maximum Modem sleep* mode;
- **MinModemAL**: *Minimum Modem sleep* mode with automatic *Light-sleep* mode;
- **MaxModemAL**: *Maximum Modem sleep* mode with automatic *Light-sleep* mode.

5.2.1 CoAP

Figure 5.1 presents the E2E latency results for the *good* signal strength channel for 85B and 850B of message payloads for each power mode and Table 5.1 indicates the mean values of the previous results. There are no packets lost in the *good* signal strength channel experiment.

Regarding the *bad* signal strength channel experiment, Figure 5.2 presents the E2E latency results for 85B and 850B of message payloads for each power mode and Table 5.2 indicates the mean values of the previous results.

For the PDR, Figure 5.3 presents the E2E latency results for 85B and 850B of message payloads for each power mode and Table 5.3 indicates the mean values of the previous results.

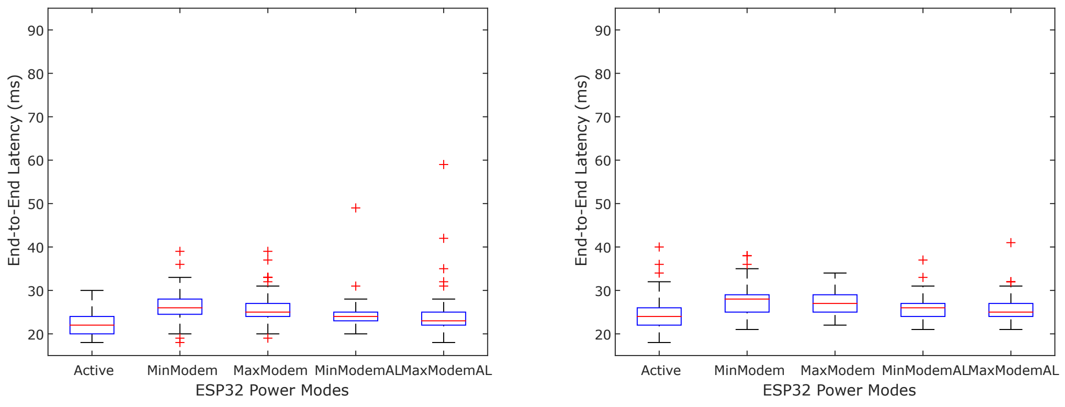


Figure 5.1: E2E latency results for the *good* signal strength channel with CoAP Non-confirmable with 85B (left) and 850B (right) of message payload for each power mode

Table 5.1: Mean values for the *good* signal strength channel experiment with CoAP Non-confirmable

Power Mode	**85B payload (ms)**	**850B payload (ms)**
Active	22.5	24.6
MinModem	26.2	28.6
MaxModem	25.9	27.1
MinModemAL	24.0	25.7
MaxModemAL	24.1	25.9
Average	24.5	26.4

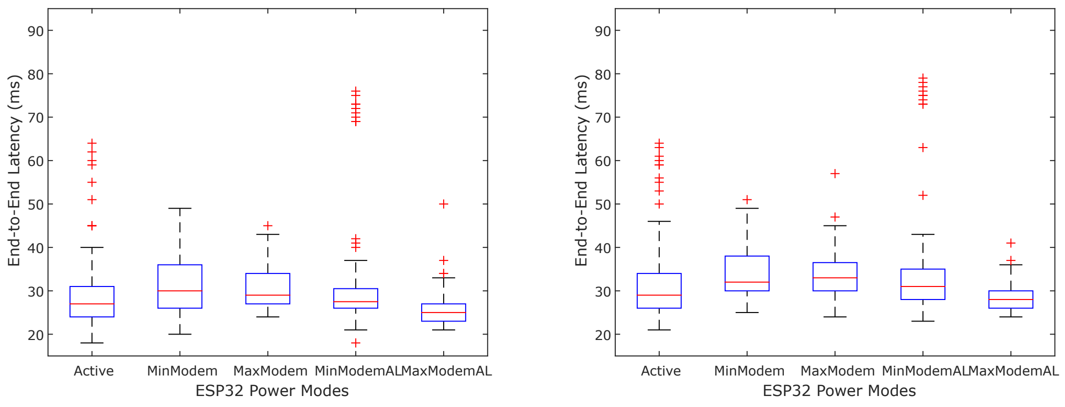


Figure 5.2: E2E latency results for the *bad* signal strength channel with CoAP Non-confirmable with 85B (left) and 850B (right) of message payload for each power mode

5.2.2 HTTP

Figure 5.4 presents the E2E latency results for the *good* signal strength channel for 85B and 850B of message payloads for each power mode and Table 5.4 indicates the mean values of the previous results. There are no packets lost for this experiment.

Regarding the *bad* signal strength channel experiment, Figure 5.5 presents the E2E latency

Table 5.2: Mean values for the *bad* signal strength channel experiment with CoAP Non-confirmable

Power Mode	**85B payload (ms)**	**850B payload (ms)**
Active	31.9	33.0
MinModem	31.6	34.0
MaxModem	30.7	33.7
MinModemAL	33.1	37.6
MaxModemAL	26.2	28.7
Average	30.7	33.4

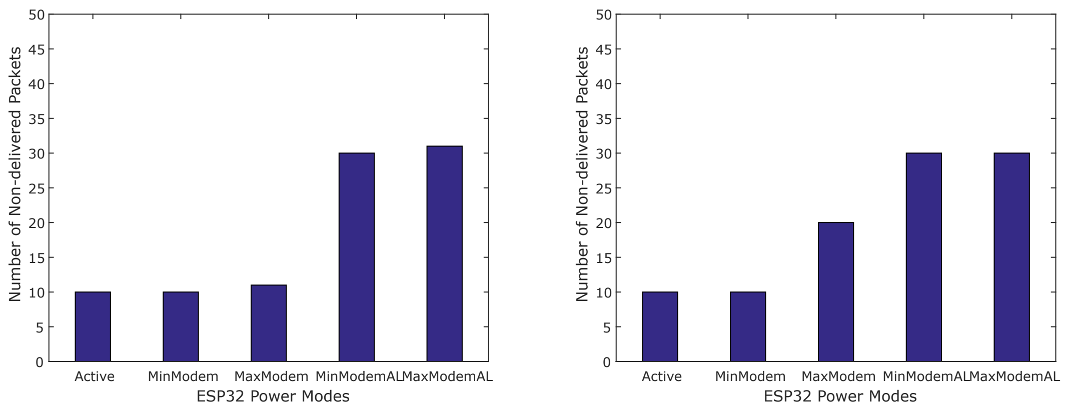


Figure 5.3: Packets Non-delivered for the *bad* signal strength channel with CoAP Non-confirmable with 85B (left) and 850B (right) of message payload for each power mode

Table 5.3: Packets Non-delivered for the *bad* signal strength channel experiment with CoAP Non-confirmable

Power Mode	**85B payload (%)**	**850B payload (%)**
Active	90	90
MinModem	90	90
MaxModem	89	80
MinModemAL	70	70
MaxModemAL	69	70
Average	81.6	80

Table 5.4: Mean values for the *good* signal strength channel experiment with HTTP

Power Mode	**85B payload (ms)**	**850B payload (ms)**
Active	22.9	24.8
MinModem	24.9	27.1
MaxModem	24.7	27.0
MinModemAL	23.1	25.3
MaxModemAL	23.2	25.4
Average	23.7	25.9

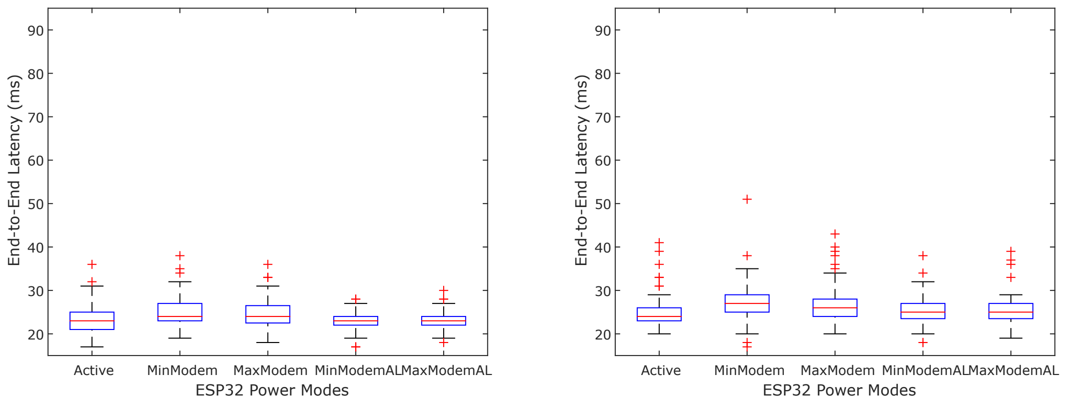


Figure 5.4: E2E latency results for the *good* signal strength channel with HTTP with 85B (left) and 850B (right) of message payload for each power mode

results for 85B and 850B of message payloads for each power mode and Table 5.5 indicates the mean values of the previous results.

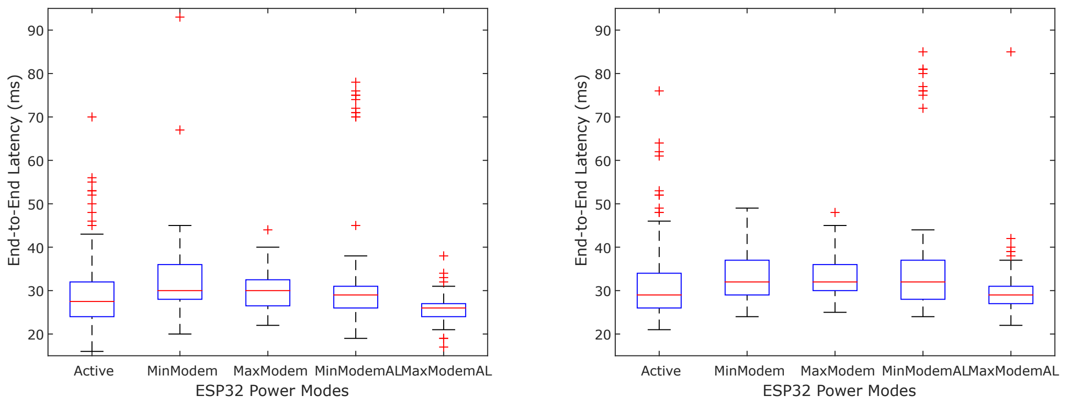


Figure 5.5: E2E latency results for the *bad* signal strength channel with HTTP with 85B (left) and 850B (right) of message payload for each power mode

Table 5.5: Mean values for the *bad* signal strength channel experiment with HTTP

Power Mode	**85B payload (ms)**	**850B payload (ms)**
Active	32.8	32.4
MinModem	30.0	33.2
MaxModem	30.1	33.8
MinModemAL	34.3	38.9
MaxModemAL	25.9	33.1
Average	30.6	34.3

For the PDR, Figure 5.6 presents the E2E latency results for 85B and 850B of message payloads for each power mode and Table 5.6 indicates the mean values of the previous results.

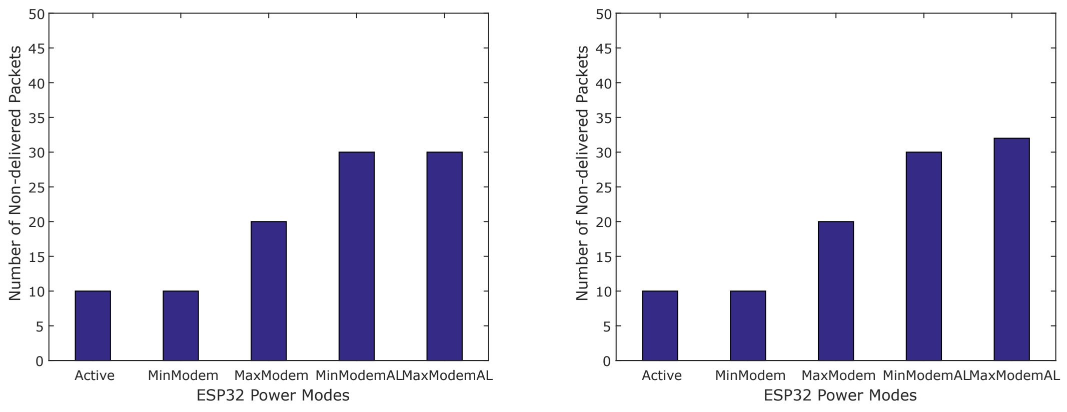


Figure 5.6: Packets Non-delivered for the *bad* signal strength channel with HTTP with 85B (left) and 850B (right) of message payload for each power mode

Table 5.6: Packets Non-delivered for the *bad* signal strength channel experiment with HTTP

Power Mode	**85B payload (%)**	**850B payload (%)**
Active	90	90
MinModem	90	90
MaxModem	80	80
MinModemAL	70	70
MaxModemAL	70	68
Average	80	79.6

5.2.3 MQTT

5.2.3.1 MQTT QoS0

Figure 5.7 presents the E2E latency results for the *good* signal strength channel for 85B and 850B of message payloads for each power mode and Table 5.7 indicates the mean values of the previous results.

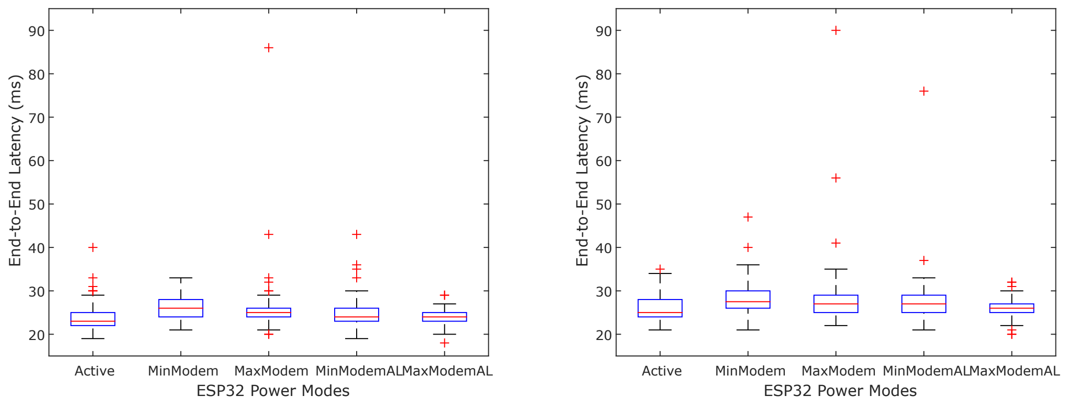


Figure 5.7: E2E latency results for the *good* signal strength channel with MQTT QoS0 with 85B (left) and 850B (right) of message payload for each power mode

Table 5.7: Mean values for the *good* signal strength channel experiment with MQTT QoS0

Power Mode	**85B payload (ms)**	**850B payload (ms)**
Active	24.0	25.9
MinModem	26.0	28.3
MaxModem	25.9	27.9
MinModemAL	24.9	27.5
MaxModemAL	24.1	26.1
Average	25.0	27.1

Regarding the *bad* signal strength channel experiment, Figure 5.8 presents the E2E latency results for 85B and 850B of message payloads for each power mode and Table 5.8 indicates the mean values of the previous results.

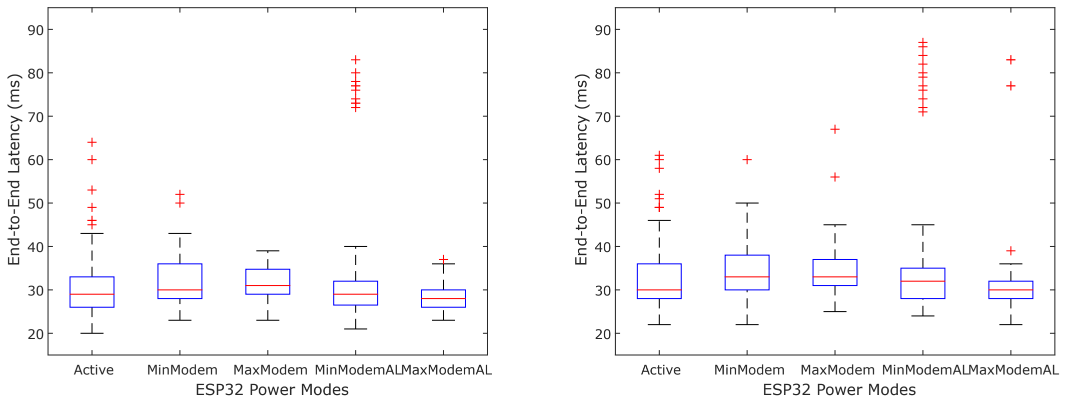


Figure 5.8: E2E latency results for the *bad* signal strength channel with MQTT QoS0 with 85B (left) and 850B (right) of message payload for each power mode

Table 5.8: Mean values for the *bad* signal strength channel experiment with MQTT QoS0

Power Mode	**85B payload (ms)**	**850B payload (ms)**
Active	31.9	34.9
MinModem	32.1	34.5
MaxModem	33.7	35.2
MinModemAL	34.6	37.7
MaxModemAL	32.9	38.2
Average	33.0	36.1

For the PDR, Figure 5.9 presents the E2E latency results for 85B and 850B of message payloads for each power mode and Table 5.9 indicates the mean values of the previous results.

5.2.3.2 MQTT QoS1

Figure 5.10 presents the E2E latency results for the *good* signal strength channel for 85B and 850B of message payloads for each power mode and Table 5.10 indicates the mean values of the previous results.

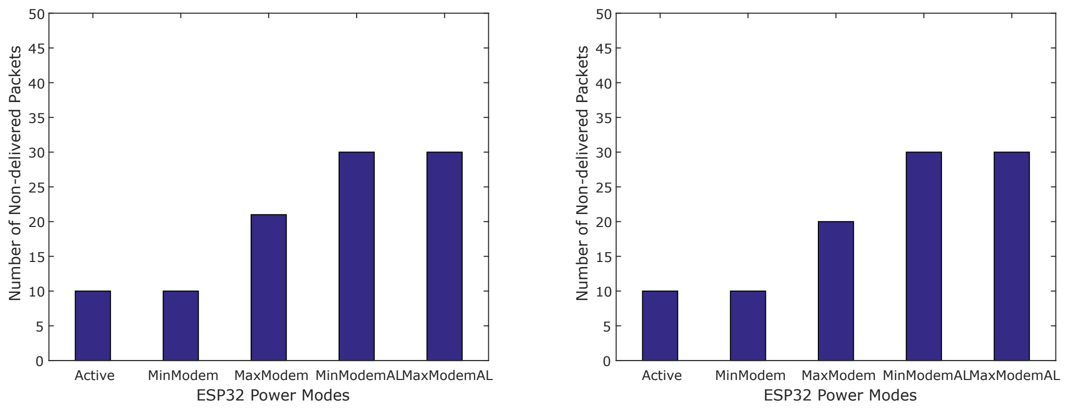


Figure 5.9: Packets Non-delivered for the *bad* signal strength channel with MQTT QoS0 with 85B (left) and 850B (right) of message payload for each power mode

Table 5.9: Packets Non-delivered for the *bad* signal strength channel experiment with MQTT QoS0

Power Mode	**85B payload (%)**	**850B payload (%)**
Active	90	90
MinModem	90	90
MaxModem	79	80
MinModemAL	70	70
MaxModemAL	70	70
Average	79.8	80

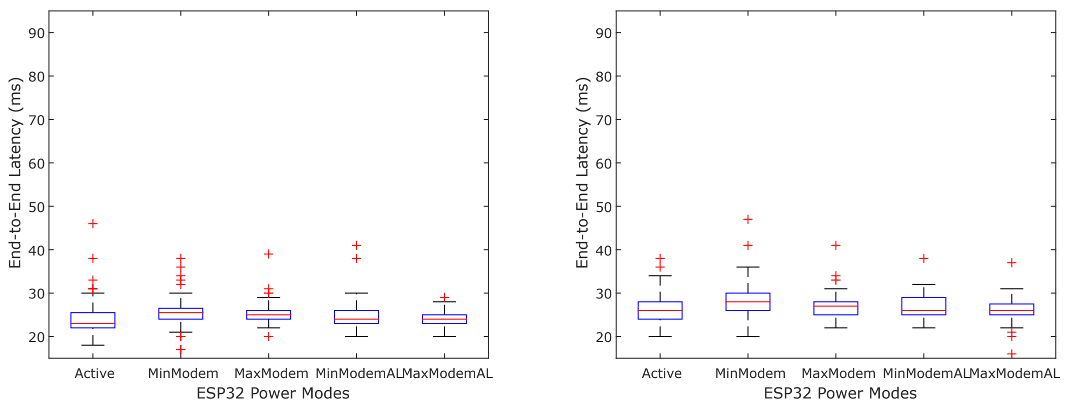


Figure 5.10: E2E latency results for the *good* signal strength channel with MQTT QoS1 with 85B (left) and 850B (right) of message payload for each power mode

Regarding the *bad* signal strength channel experiment, Figure 5.11 presents the E2E latency results for 85B and 850B of message payloads for each power mode and Table 5.11 indicates the mean values of the previous results.

For the PDR, Figure 5.12 presents the E2E latency results for 85B and 850B of message payloads for each power mode and Table 5.12 indicates the mean values of the previous results.

Table 5.10: Mean values for the *good* signal strength channel experiment with MQTT QoS1

Power Mode	**85B payload (ms)**	**850B payload (ms)**
Active	24.2	26.1
MinModem	25.6	28.1
MaxModem	25.2	27.1
MinModemAL	26.0	26.7
MaxModemAL	26.9	26.3
Average	25.6	26.7

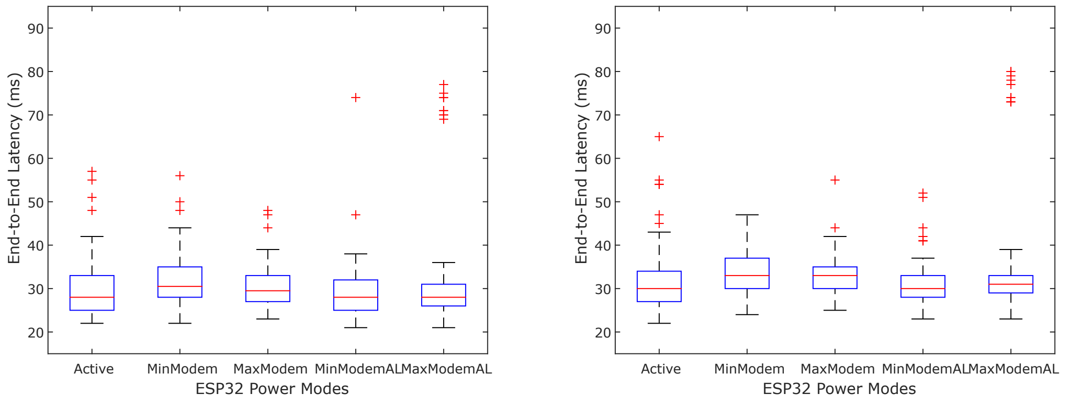


Figure 5.11: E2E latency results for the *bad* signal strength channel with MQTT QoS1 with 85B (left) and 850B (right) of message payload for each power mode

Table 5.11: Mean values for the *bad* signal strength channel experiment with MQTT QoS1

Power Mode	**85B payload (ms)**	**850B payload (ms)**
Active	30.1	31.8
MinModem	32.2	33.6
MaxModem	35.4	36.0
MinModemAL	29.4	31.1
MaxModemAL	34.1	36.8
Average	32.2	33.9

Table 5.12: Packets Non-delivered for the *bad* signal strength channel experiment with MQTT QoS1

Power Mode	**85B payload (%)**	**850B payload (%)**
Active	90	90
MinModem	90	90
MaxModem	80	80
MinModemAL	70	70
MaxModemAL	70	70
Average	80	80

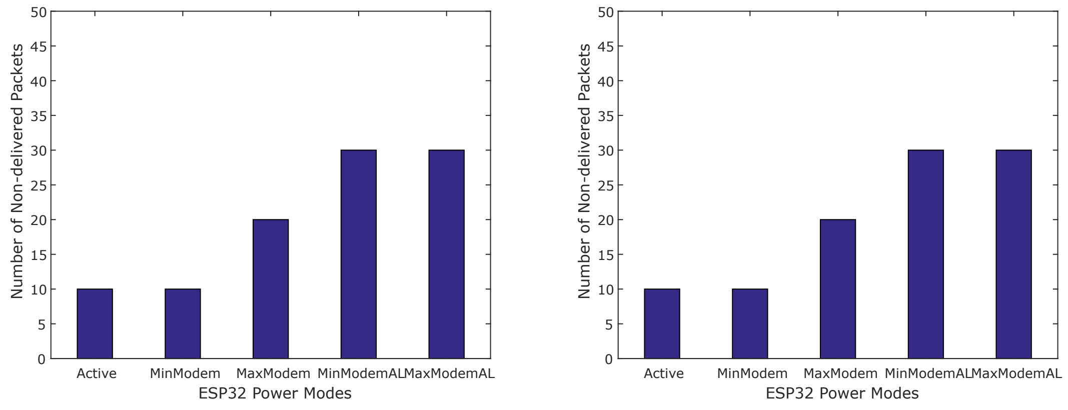


Figure 5.12: Packets Non-delivered for the *bad* signal strength channel with MQTT QoS1 with 85B (left) and 850B (right) of message payload for each power mode

5.2.3.3 MQTT QoS2

Figure 5.13 presents the E2E latency results for the *good* signal strength channel for 85B and 850B of message payloads for each power mode and Table 5.13 indicates the mean values of the previous results.

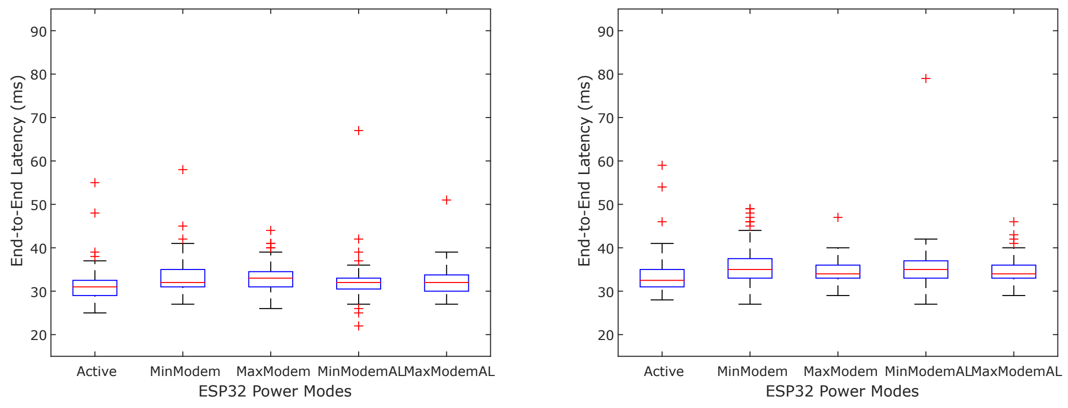


Figure 5.13: E2E latency results for the *good* signal strength channel with MQTT QoS2 with 85B (left) and 850B (right) of message payload for each power mode

Table 5.13: Mean values for the *good* signal strength channel experiment with MQTT QoS2

Power Mode	85B payload (ms)	850B payload (ms)
Active	31.4	33.4
MinModem	33.2	35.7
MaxModem	33.0	34.6
MinModemAL	32.4	37.4
MaxModemAL	33.8	39.6
Average	32.8	36.1

Regarding the *bad* signal strength channel experiment, Figure 5.14 presents the E2E latency results for 85B and 850B of message payloads for each power mode and Table 5.14 indicates the mean values of the previous results.

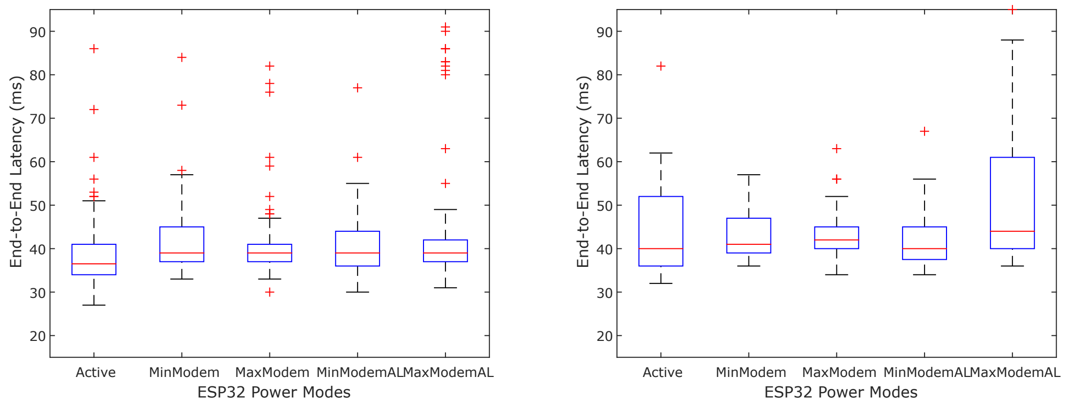


Figure 5.14: E2E latency results for the *bad* signal strength channel with MQTT QoS2 with 85B (left) and 850B (right) of message payload for each power mode

Table 5.14: Mean values for the *bad* signal strength channel experiment with MQTT QoS2

Power Mode	**85B payload (ms)**	**850B payload (ms)**
Active	43.8	56.8
MinModem	44.0	43.8
MaxModem	41.4	66.4
MinModemAL	49.2	55.4
MaxModemAL	45.9	72.7
Average	44.9	59.0

For the PDR, Figure 5.15 presents the E2E latency results for 85B and 850B of message payloads for each power mode and Table 5.15 indicates the mean values of the previous results.

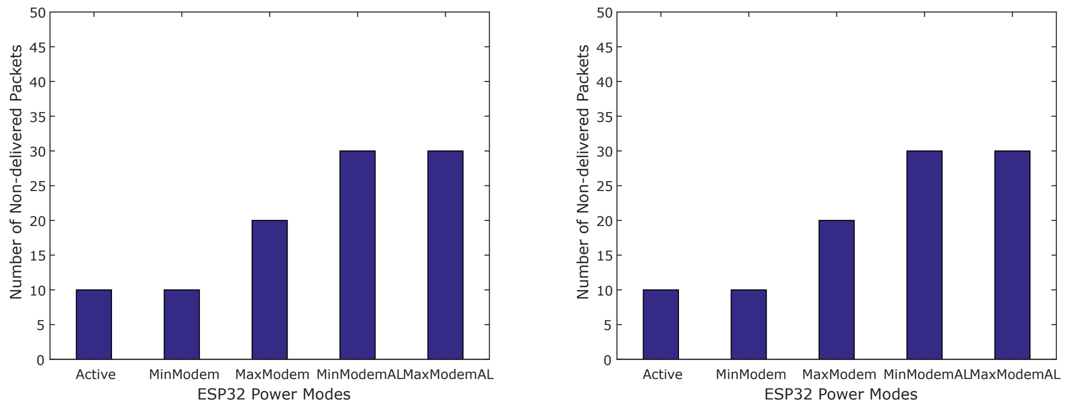


Figure 5.15: Packets Non-delivered for the *bad* signal strength channel with MQTT QoS2 with 85B (left) and 850B (right) of message payload for each power mode

Table 5.15: Packets Non-delivered for the *bad* signal strength channel experiment with MQTT QoS2

Power Mode	**85B payload (%)**	**850B payload (%)**
Active	90	90
MinModem	90	90
MaxModem	80	80
MinModemAL	70	70
MaxModemAL	70	70
Average	80	80

5.2.4 Protocols Comparison

Figure 5.16 provides an overview the impact of all the protocols and ESP32 module's power modes on E2E latency with 85B of payload for the *good* signal strength channel experiments. Additionally, Figure 5.17 presents the E2E latency results with 850B of payload for the *good* signal strength channel experiment. As stressed earlier, there are no packets lost for this experiment

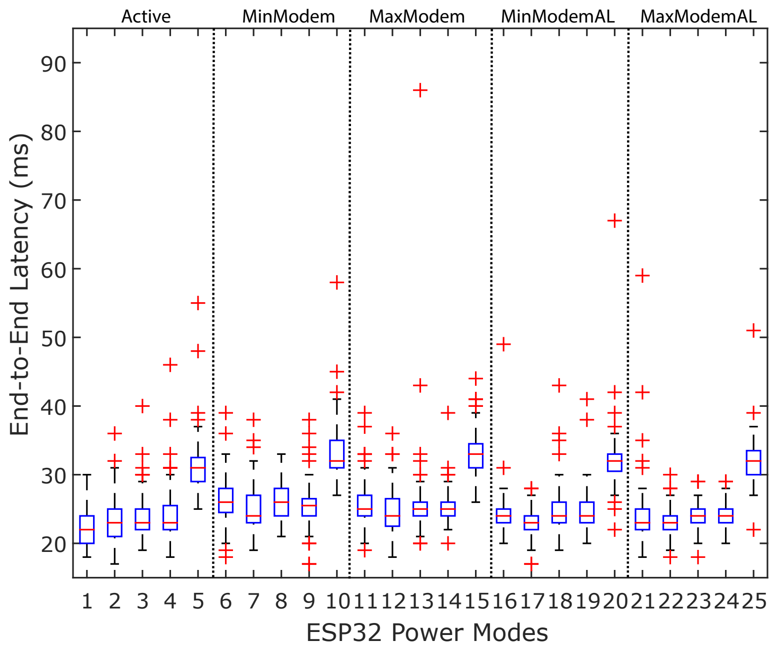


Figure 5.16: E2E latency results with 85B of payload for the *good* signal strength channel experiment

Regarding the *bad* signal strength channel experiment, Figures 5.18 and 5.19 present the E2E latency results with 85B and 850B of payload, respectively.

For the PDR, Figure 5.20 and Figure 5.21 show the packets non-delivered for messages with 85B and 850B for the "bad" channel, respectively.

These graphics are composed by blocks of five measurements. The first measurements, 1-5, correspond to the Active mode. Measurements 6-10 correspond to Minimum Modem mode and

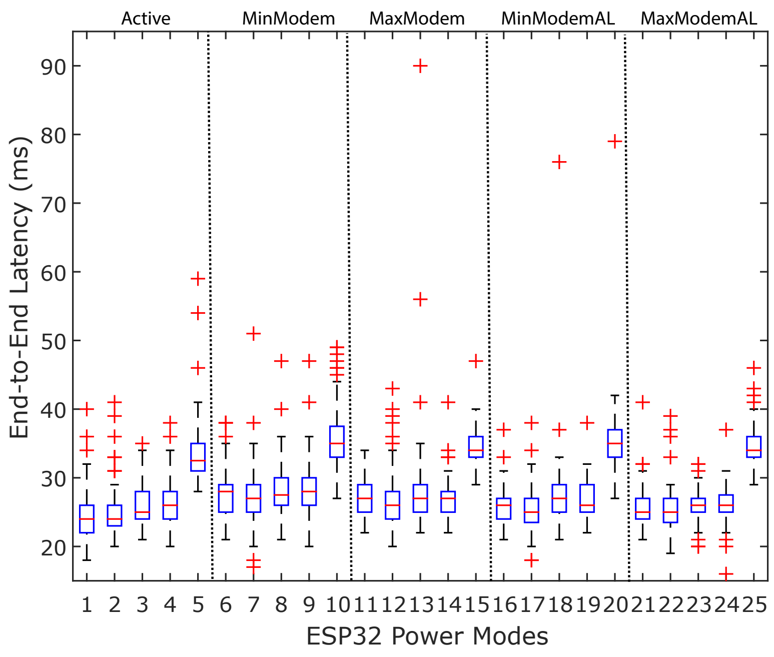


Figure 5.17: E2E latency results with 850B of payload for the *good* signal strength channel experiment

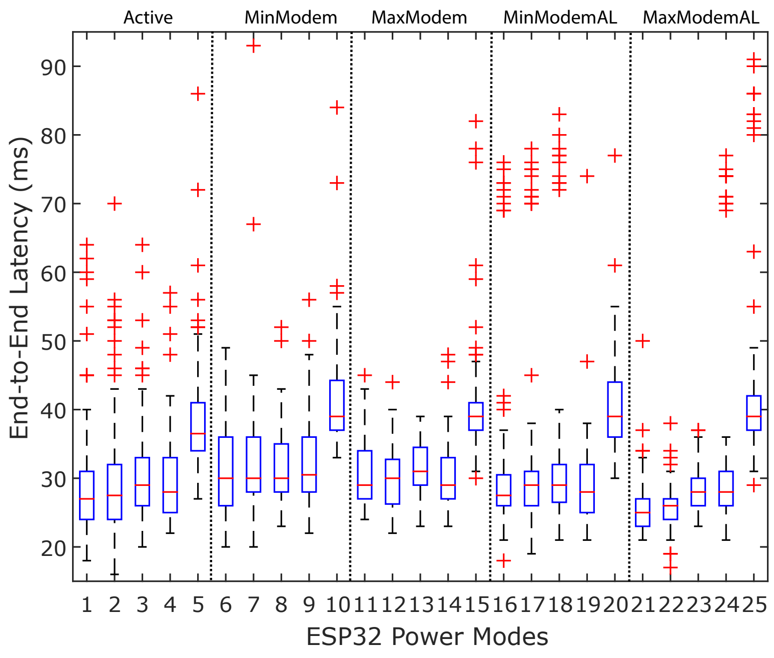


Figure 5.18: E2E latency results with 85B of payload for the *bad* signal strength channel experiment

measurements 11-15 correspond to Maximum Modem. The last two blocks are associated the Minimum Modem and Maximum Modem modes with automatic Light sleep, respectively. Inside the block, the positions are organised by protocols:

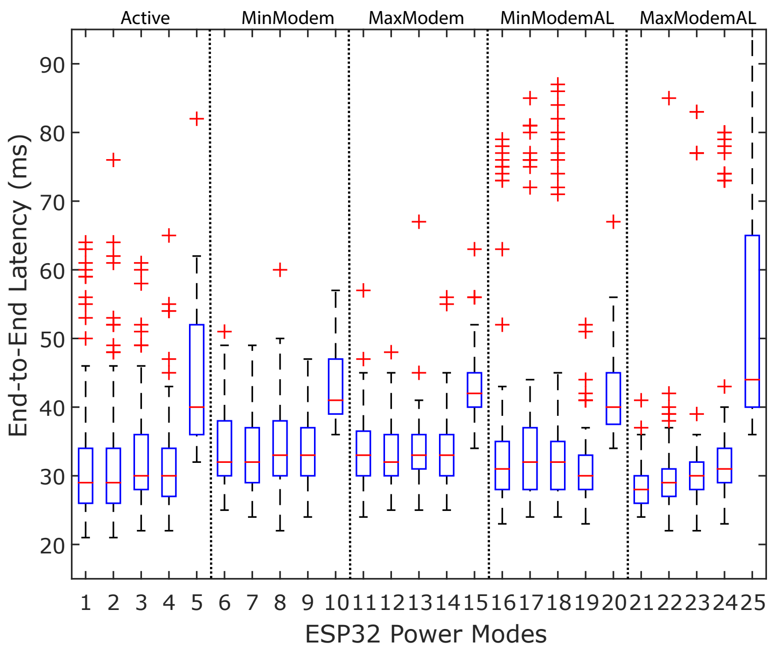


Figure 5.19: E2E latency results with 850B of payload for the *bad* signal strength channel experiment

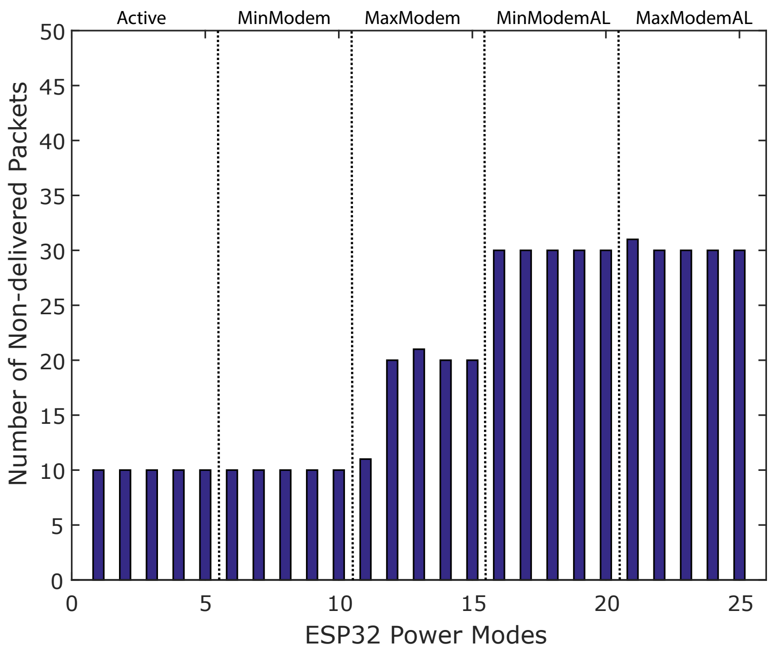


Figure 5.20: Packets non-delivered with 85B of payload for the *bad* signal strength channel experiment

- Position 1: CoAP Non-confirmable;
- Position 2: HTTP;

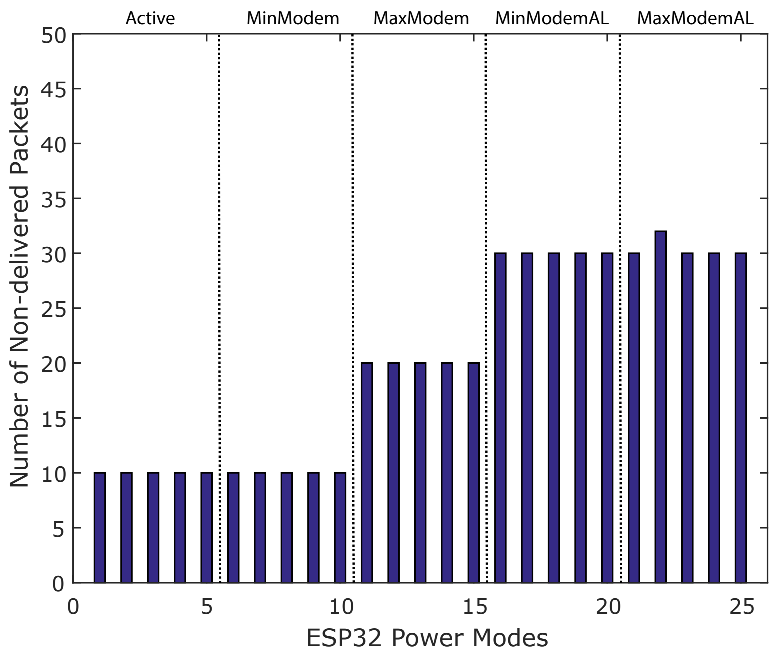


Figure 5.21: Packets non-delivered with 850B of payload for the *bad* signal strength channel experiment

- Position 3: MQTT QoS0;
- Position 4: MQTT QoS1;
- Position 5: MQTT QoS2.

As expected, the E2E latency results for the *bad* signal strength channel experiment are greater than the results for the *good* signal strength channel as it has an average increase of 30.24% for 85B and 38.20% for 850B.

The different message payload also has an impact on the E2E latency results as the results for the 850B of payload increase when compared to the 85B of payload results. For the *good* signal strength channel experiment, there is an increase of 7.42% for CoAP Non-confirmable, 9.18% for HTTP, 8.56% for MQTT QoS0, 5.00% for MQTT QoS1 and 10.25% for MQTT QoS2 for 850B of payload. Thus, for the *good* signal strength channel, there's is an average increase of 8.08%. Regarding the *bad* signal strength channel there is also an increase on E2E latency when comparing the results 850B to 85B of payload for all protocols. The average latency increases 8.82% for CoAP Non-confirmable, 12.05% for HTTP, 9.39% for MQTT QoS0, 5.03% for MQTT QoS1 and 31.58% for MQTT QoS2. Moreover, there is an average increase of 38.14% for 850B when comparing the two channels experiments.

For the *good* signal strength channel, the E2E latency experienced is within 20-28ms for messages with 85B of payload for all protcols except MQTT QoS2, which experience E2E latency between 29-35ms.

The protocols that use TCP as its transport layer protocol, i.e. MQTT and HTTP, shown an increased latency relatively to CoAP uses the UDP, although this differences are negligible for the *good* signal strength channel experiment, except for MQTT QoS2 (Table 5.13).

Regarding the ESP32 module's power modes, a greater latency is experienced when comparing more energy-efficient power modes with the less energy-efficient power modes, although it is not significant and in some cases the latency is bigger in less energy-efficient power modes. For example, the E2E latency decreases 26.26% for Maximum Modem mode with Automatic Light Sleep when compared to Active mode for CoAP Non-confirmable with 85B payload for the *bad* signal strength channel experiment.

For the *good* signal strength channel, the PDR is 100% although that is not the case for the *bad* signal strength channel, as expected. We conclude that the value is higher in less energy-efficient power modes when comparing with more energy-efficient power modes for the *bad* signal strength channel, regardless of message payload and ALP. As shown in Figure 5.20 and Figure 5.21, the differences in the PDR between messages with 85B and 850B and protocols are negligible. When comparing the ESP32 power modes, we conclude that there is decrease of 10% for the *Maximum Modem sleep* and 20% for both *Minimum Modem sleep* and *Maximum Modem sleep* with automatic *Light-sleep* on the PDR when comparing to the *Active-sleep* and *Minimum Modem sleep*.

To conclude, the WiFi channel quality has a great impact on the experienced E2E latency. The TCP-based protocols also experience a greater latency when compared to CoAP. MQTT QoS2 is the protocol that experiences a greater E2E latency due to the fact that exchanges four messages to publish one message. Finally, scenarios where using more energy-efficient power modes experience more E2E latency and less PDR.

5.3 Summary

In this chapter, we present the methodology and the results for system's performance evaluation. The results validate the use of the ESP32 module on applications similar to the one proposed in this project, by publishing messages with 85B and 850B of payload with 1 second and 10 seconds period for different WiFi quality channels. The results show that the latency experience when comparing the channel qualities is significant, however when comparing the latency experienced with different power modes in the same channel quality scenario is not significant. Finally, the PDR is higher when the ESP32 is using a less energy-efficient power mode when comparing to a more energy-efficient power mode.

Chapter 6

Conclusion

IoT technologies have experienced a rapid growth in recent years in their applicability in several domains, such as the healthcare. Therefore, several standards and protocols were developed specifically for this new technology. In today's society, WiFi is an omnipresent technology, thus small embedded systems start supporting WiFi technology. This is a key feature due to avoidance of a GW device connect the system to the Internet.

In this work, we proposed and implemented an IoT system that envisions to satisfy the requirements of remote health monitoring while using a low-cost low-cost WiFi-enabled device. To guarantee the interoperability between devices, we used the oneM2M standard. Regarding M2M communications between devices, we implemented CoAP, HTTP and MQTT protocols.

The system proposed is based on the oneM2M, which allows a publisher-subscriber communication model, a great model for sensing and remote monitoring application such as this. The system proposed comprises a wearable based on an Espressif ESP32 module, a MAX30102 PPG module and a LiPo battery, and a monitoring application with GUI as well as a database. InfluxDB, and a data visualisation tool. The ESP32 module is the central piece of the wearable because processes the data acquired in the MAX30102 module, i.e. HR and S_pO_2, and calculates battery percentage and publishes to the OM2M broker, IN-CSE. The monitoring application controls the data flow between the main components of the system, thus communicating with the InfluxDB and subscribing data from the IN-CSE. We also developed a GUI to give a visually aid of the real-time status of the system as well as controlling it.

Through the GUI, the medical personnel can add a wearable(s) to the system which allows the wearable(s) to start publishing data. At any moment, the medical personnel can delete the wearable(s) from the system as well as pause the wearable. Additionally, the GUI offers a button to open the Chronograf application which enables the medical personnel to analyse data stored on the database.

Regarding the system's performance, we conducted E2E latency experiments, with data transmissions at different frequencies, using different power modes offered by the ESP32 module for two WiFi channel quality experiments regarding signal strength. The results show that the E2E latency for a *bad* WiFi channel quality increases 30.24% and 38.20% for messages published with

1 second and 10 seconds periods, respectively, when compared to a *good* WiFi channel. We also conclude that scenarios with TCP-based protocols,such as the HTTP and MQTT, experience more E2E delay than UDP-based protocols, i.e. CoAP. Furthermore, for the 850B of payload measures, there is an increase of 26.26% when comparing to 85B of payload for the two channels experiments. Moreover, we conclude that the PDR is 100% in the *good* signal strength channel experiment while in the *bad* signal strength experiment there is a decrease of the PDR when the ESP32 is using the more energy-efficient power modes.

6.1 Future Work

The work done can be continued and improved in different facets. Namely, conduct experiments to determine current consumption for different scenarios, and consequently, battery life for the proposed architecture. Also regarding the battery, conduct an in-depth study of the various options.

Wearable design and, consequently, build would be an important task as in this dissertation we prototype the device in a breadboard. Moreover, testing the system in real-life environment, e.g. in an sports competition game, would also be an important task.

Lastly, improve the GUI to display the real-time data graphics, connected to the Chronograf if possible, so that the medical personnel would not need to open a browser page and configure the Chronograf dashboard.

References

[1] World Health Organization. Life expectancy, 2020. https://www.who.int/gho/mortality_burden_disease/life_tables/situation_trends_text/en/, Last accessed on 2020-06-25.

[2] World Health Organization. Cardiovascular diseases, 2020. https://www.who.int/health-topics/cardiovascular-diseases/#tab=tab_1, Last accessed on 2020-06-25.

[3] Carlos Pereira, João Mesquita, Diana Guimarães, Frederico Santos, Luis Almeida, and Ana Aguiar. Open IoT Architecture for Continuous Patient Monitoring in Emergency Wards. *Electronics (Switzerland)*, 8(10):1–15, 2019. doi:10.3390/electronics8101074.

[4] Open Connectivity Foundation. Alljoyn, 2020. https://openconnectivity.org/technology/reference-implementation/alljoyn/, Last accessed on 2020-06-14.

[5] Anum Ali, Ghalib A. Shah, Muhammad Omer Farooq, and Usman Ghani. Technologies and challenges in developing Machine-to-Machine applications: A survey. *Journal of Network and Computer Applications*, 83(September 2016):124–139, 2017. URL: http://dx.doi.org/10.1016/j.jnca.2017.02.002, doi:10.1016/j.jnca.2017.02.002.

[6] AllJoyn. Architecture, 2015. https://github.com/alljoyn/extras-webdocs/blob/master/docs/learn/architecture.md, Last accessed on 2020-02-10.

[7] FIWARE. Fiware: The open source platform for our smart digital future, 2020. https://www.fiware.org, Last accessed on 2020-01-30.

[8] FIWARE. What is fiware?, 2020. https://www.fiware.org, Last accessed on 2020-01-30.

[9] Jahoon Koo, Se Ra Oh, and Young Gab Kim. Device identification interoperability in heterogeneous iot platforms †. *Sensors (Switzerland)*, 19(6), 2019. doi:10.3390/s19061433.

[10] FIWARE. Fiware catalogue, 2020. https://www.fiware.org/developers/catalogue/, Last accessed on 2020-02-10.

[11] IoTivity. Home | iotivity, 2020. https://iotivity.org, Last accessed on 2020-01-30.

[12] Jaehong Jo, Jaehyun Cho, Rami Jung, and Hanna Cha. IoTivity-Lite: Comprehensive IoT Solution in A Constrained Memory Device. *9th International Conference on Information and Communication Technology Convergence: ICT Convergence Powered by Smart Intelligence, ICTC 2018*, pages 1367–1369, 2018. doi:10.1109/ICTC.2018.8539598.

[13] OpenIoT. Open source cloud solution for the internet of things, 2020. http://www.openiot.eu, Last accessed on 2020-01-30.

[14] OpenIoT. Openiot architecture, 2013. https://github.com/OpenIotOrg/openiot/wiki/OpenIoT-Architecture, Last accessed on 2020-02-10.

[15] IERC. Ierc - european research cluster on the internet of things, 2020. http://www.internet-of-things-research.eu, Last accessed on 2020-02-09.

[16] oneM2M. Standards for m2m and the internet of things, 2020. http://www.onem2m.org, Last accessed on 2020-01-30.

[17] Jörg Swetina, Guang Lu, Philip Jacobs, Francois Ennesser, and Jaeseung Song. Toward a standardized common M2M service layer platform: Introduction to oneM2M. *IEEE Wireless Communications*, 21(3):20–26, 2014. doi:10.1109/MWC.2014.6845045.

[18] oneM2M. Functional architecture description, 2020. http://www.onem2m.org/getting-started/onem2m-overview/introduction/functional-architecture, Last accessed on 2020-01-30.

[19] oneM2M. onem2m service layer, 2020. http://www.onem2m.org/getting-started/onem2m-overview/introduction/service-layer, Last accessed on 2020-01-30.

[20] Eclipse Foundation. What is om2m?, 2020. https://www.eclipse.org/om2m/, Last accessed on 2020-01-30.

[21] L. Zilhao, Ricardo Morla, and Ana Aguiar. A Modular Tool for Benchmarking loT Publish-Subscribe Middleware. *19th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM 2018*, 2018. doi:10.1109/WoWMoM.2018.8449774.

[22] M. Ben Alaya, Y. Banouar, T. Monteil, C. Chassot, and K. Drira. OM2M: Extensible ETSI-compliant M2M service platform with self-configuration capability. *Procedia Computer Science*, 32:1079–1086, 2014. URL: http://dx.doi.org/10.1016/j.procs.2014.05.536, doi:10.1016/j.procs.2014.05.536.

[23] Carlos Pereira, João Cardoso, Ana Aguiar, and Ricardo Morla. Benchmarking Pub/Sub IoT middleware platforms for smart services. *Journal of Reliable Intelligent Environments*, 4(1):25–37, 2018. URL: https://doi.org/10.1007/s40860-018-0056-3, doi:10.1007/s40860-018-0056-3.

[24] Charilaos Akasiadis, Vassilis Pitsilis, and Constantine D. Spyropoulos. A multi-protocol IoT platform based on open-source frameworks. *Sensors (Switzerland)*, 19(19):1–25, 2019. doi:10.3390/s19194217.

[25] Sierra Wireless. Welcome to the source, 2020. https://source.sierrawireless.com/#sthash.yjBWRUsl.DhXX9D8Z.dpbs, Last accessed on 2020-01-30.

[26] Sierra Wireless Legato. Build platform, 2020. https://docs.legato.io/latest/buildPlatformMain.html, Last accessed on 2020-02-10.

[27] Open MTC. Open mtc, 2020. https://www.openmtc.org, Last accessed on 2020-02-10.

[28] Andrew (PrismTech) Foster. Messaging Technologies for the Industrial Internet and the Internet of Things Whitepaper. *Prismtech*, (March):1–22, 2014. URL: http://www.prismtech.com/sites/default/files/documents/MessagingComparsionMarch2014USROW-final.pdf.

[29] Nitin Naik. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. *2017 IEEE International Symposium on Systems Engineering, ISSE 2017 - Proceedings*, 2017. doi:10.1109/SysEng.2017.8088251.

[30] OASIS. Advanced message queuing protocol (amqp) version 1.0, 2012. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-transport-v1.0-os.html, Last accessed on 2020-06-15.

[31] Yuang Chen and Thomas Kunz. Performance Evaluation of IoT Protocols under a Constrained Wireless Access Network. *2016 International Conference on Selected Topics in Mobile and Wireless Networking, MoWNeT 2016*, pages 1–7, 2016. doi:10.1109/MoWNet.2016.7496622.

[32] Burak H. Çorak, Feyza Y. Okay, Metehan Güzel, Şahin Murt, and Suat Ozdemir. Comparative Analysis of IoT Communication Protocols. *2018 International Symposium on Networks, Computers and Communications, ISNCC 2018*, 2018. doi:10.1109/ISNCC.2018.8530963.

[33] Carsten Bormann, Angelo P. Castellani, and Zach Shelby. CoAP: An application protocol for billions of tiny internet nodes. *IEEE Internet Computing*, 16(2):62–67, 2012. doi:10.1109/MIC.2012.29.

[34] IETF. The constrained application protocol (coap), 2014. https://tools.ietf.org/html/rfc7252, Last accessed on 2020-06-15.

[35] Soma Bandyopadhyay and Abhijan Bhattacharyya. Lightweight Internet protocols for web enablement of sensors using constrained gateway devices. *2013 International Conference on Computing, Networking and Communications, ICNC 2013*, pages 334–340, 2013. doi:10.1109/ICCNC.2013.6504105.

[36] Konstantinos Fysarakis, Ioannis Askoxylakis, Othonas Soultatos, Ioannis Papaefstathiou, Charalampos Manifavas, and Vasilios Katos. Which IoT Protocol? Comparing standardized approaches over a common M2M application. *2016 IEEE Global Communications Conference (GLOBECOM)*, 2016. URL: http://dx.doi.org/10.1109/GLOCOM.2016.7842383VN-readcube.com, doi:10.1109/GLOCOM.2016.7842383.

[37] OASIS. Messaging queueing telemetry transport (mqtt), 2019. https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html, Last accessed on 2020-06-15.

[38] Paridhika Kayal and Harry Perros. A comparison of IoT application layer protocols through a smart parking implementation. *Proceedings of the 2017 20th Conference on Innovations in Clouds, Internet and Networks, ICIN 2017*, pages 331–336, 2017. doi:10.1109/ICIN.2017.7899436.

[39] I. Hedi, I. Špeh, and A. Šarabok. IoT network protocols comparison for the purpose of IoT constrained networks. *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2017 - Proceedings*, pages 501–505, 2017. doi:10.23919/MIPRO.2017.7973477.

[40] Matthias Pohl, Janick Kubela, Sascha Bosse, and Klaus Turowski. Performance evaluation of application layer protocols for the internet-of-things. *Proceedings - 2018 6th International Conference on Enterprise Systems, ES 2018*, pages 180–187, 2018. doi:10.1109/ES.2018.00035.

[41] Marko Pavelic, Vatroslav Bajt, and Mario Kusek. Energy efficiency of machine-to-machine protocols. *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2018 - Proceedings*, pages 361–366, 2018. doi:10.23919/MIPRO.2018.8400069.

[42] Bluetooth SIG. Bluetooth low energy (ble), 2020. https://www.bluetooth.com/learn-about-bluetooth/bluetooth-technology/radio-versions/, Last accessed on 2020-06-15.

[43] Joao Mesquita, Diana Guimaraes, Carlos Pereira, Frederico Santos, and Luis Almeida. Assessing the ESP8266 WiFi module for the Internet of Things. *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, 2018-Septe:784–791, 2018. doi:10.1109/ETFA.2018.8502562.

[44] Espressif Systems. ESP32 Series Datasheet. *Espressif Systems*, pages 1–61, 2019. URL: https://www.espressif.com/sites/default/files/documentation/esp32{_}datasheet{_}en.pdf.

[45] Yindu Building and Haidian District. W600 Specification. pages 1–21, 2019. URL: http://www.winnermicro.com/en/html/1/156/158/497.html.

[46] InfluxData. Influxdb 1.x, 2020. https://www.influxdata.com/time-series-platform/, Last accessed on 2020-06-15.

[47] InfluxData. Influxdb 1.8 documentation, 2020. https://docs.influxdata.com/influxdb/v1.8/, Last accessed on 2020-06-15.

[48] InfluxData. Chronograf, 2020. https://www.influxdata.com/time-series-platform/chronograf/, Last accessed on 2020-06-15.

[49] InfluxData. Kapacitor, 2020. https://www.influxdata.com/time-series-platform/kapacitor/, Last accessed on 2020-06-15.

[50] The Linux Foundation. Prometheus, 2020. https://prometheus.io, Last accessed on 2020-06-15.

[51] The Linux Foundation. Prometheus overview, 2020. https://prometheus.io/docs/introduction/overview/, Last accessed on 2020-06-15.

[52] Grafana Labs. Grafana, 2020. https://grafana.com, Last accessed on 2020-06-15.

[53] Grafana Labs. Grafana features, 2020. https://grafana.com/grafana/, Last accessed on 2020-06-15.

[54] Thinger.io. Thinger.io, 2020. https://thinger.io, Last accessed on 2020-06-15.

[55] Thinger.io. Thinger.io overview, 2020. https://docs.thinger.io, Last accessed on 2020-06-15.

[56] Lawrence Oriaghe Aghenta and Mohammad Tariq Iqbal. Low-Cost, Open Source IoT-Based SCADA System Design Using Thinger.IO and ESP32 Thing. *Electronics (Switzerland)*, 8(8):1–24, 2019. doi:10.3390/electronics8080822.

[57] Anil Yadav, Nitin Rakesh, Sujata Pandey, and Rajat K. Singh. Development and analysis of IoT framework for healthcare application. *Advances in Intelligent Systems and Computing*, 554:149–158, 2018. doi:10.1007/978-981-10-3773-3_15.

[58] Juan M. Santos-Gago, Mateo Ramos-Merino, Sonia Vallarades-Rodriguez, Luis M. Álvarez-Sabucedo, Manuel J. Fernández-Iglesias, and Jose L. García-Soidán. Innovative Use of Wrist-Worn Wearable Devices in the Sports Domain: A Systematic Review. *Electronics (Switzerland)*, 8(11):1–29, 2019. doi:10.3390/electronics8111257.

[59] Emily J. Walker, Andrew J. McAinch, Alice Sweeting, and Robert J. Aughey. Inertial sensors to estimate the energy expenditure of team-sport athletes. *Journal of Science and Medicine in Sport*, 19(2):177–181, 2016. URL: http://dx.doi.org/10.1016/j.jsams.2015.01.013, doi:10.1016/j.jsams.2015.01.013.

[60] Marko Kos and Iztok Kramberger. A Wearable Device and System for Movement and Biometric Data Acquisition for Sports Applications, 2017. doi:10.1109/ACCESS.2017.2675538.

[61] Ilkka Parak, Jakub and Uuskoski, Maria and Machek, Jan and Korhonen. Estimating Heart Rate, Energy Expenditure, and Physical Performance With a Wrist Photoplethysmographic Device During Running. *JMIR Mhealth Uhealth*, 5(7):e97, 2017. URL: http://mhealth.jmir.org/2017/7/e97/https://doi.org/10.2196/mhealth.7437http://www.ncbi.nlm.nih.gov/pubmed/28743682, doi:10.2196/mhealth.7437.

[62] Keigo Enomoto, Ryosuke Shimizu, and Hiroyuki Kudo. Real-Time Skin Lactic Acid Monitoring System for Assessment of Training Intensity, 2018. doi:10.1002/ecj.12061.

[63] Abolfazl Soltani, Hooman Dejnabadi, Martin Savary, and Kamiar Aminian. Real-world gait speed estimation using wrist sensor: A personalized approach, 2019. doi:10.1109/jbhi.2019.2914940.

[64] Eclipse Foundation. Om2m/one, 2020. https://wiki.eclipse.org/OM2M/one, Last accessed on 2020-02-10.

[65] Eclipse Foundation. Eclipse mosquitto, 2020. http://projects.eclipse.org/projects/technology.mosquitto, Last accessed on 2020-06-17.

[66] Eclipse Foundation. Eclipse paho, 2020. http://projects.eclipse.org/projects/technology.mosquitto, Last accessed on 2020-06-17.

[67] Espressif Systems. Espressif iot development framework api source, 2020. https://github.com/espressif/esp-idf, Last accessed on 2020-06-17.

[68] Espressif Systems. Espressif iot development framework api reference, 2020. https://docs.espressif.com/projects/esp-idf/en/stable/api-reference/index.html, Last accessed on 2020-06-17.

[69] Espressif Systems. Wifi power save example, 2020. https://github.com/espressif/esp-idf/tree/release/v3.3/examples/wifi/power_save, Last accessed on 2020-06-17.

[70] Maxim Integrated. High-Sensitivity Pulse Oximeter and Heart-Rate Sensor for Wearable Health MAX30102. Technical report, 2015. URL: https://www.maximintegrated.com/en/products/interface/sensor-interface/MAX30102.html.

[71] InfluxData. Influxdb key concepts, 2020. https://docs.influxdata.com/influxdb/v1.8/concepts/key_concepts/, Last accessed on 2020-06-24.

[72] Miguel Ribeiro. Wearable sensor for continuous monitoring of physiological parameters. Master's thesis, Faculty of Engeneering (FEUP), University Of Porto, 2020.

[73] Maxim Integrated. Pulse Oximeter and Heart-Rate Sensor IC for Wearable Health. Technical report, 2015. URL: https://www.maximintegrated.com/en/products/sensors/MAX30100.html.

[74] oneM2M. Mqtt protocol binding, 2018. http://www.onem2m.org/images/files/deliverables/Release2A/TS-0010-MQTT_protocol_binding-v_2_7_1.pdf, Last accessed on 2020-07-01.

[75] João Mesquita. Comunicação WiFi para monitorização móvel de sinais fisiológicos. Master's thesis, Faculty of Engeneering (FEUP), University of Porto, 2018.

[76] Eclipse Foundation. Mqtt c client for posix and windows, 2020. https://www.eclipse.org/paho/clients/c/, Last accessed on 2020-07-05.

[77] Eclipse Foundation. onem2m java applications, 2017. https://wiki.eclipse.org/OM2M/one/App, Last accessed on 2020-07-01.