FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Optimization of a cloud-based biological sample data processing system

Mário Diogo Pinto da Silva Cardoso

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: João Paulo de Sousa

External Supervisor: Paulo Santos

July 30, 2021

© Mário Diogo Pinto da Silva Cardoso, 2021

Resumo

No contexto da medicina personalizada, têm sido feitos progressos no sentido da integração da Fotónica e da Inteligência Artificial na criação de uma biblioteca virtual de biomarcadores de doenças e perfis biológicos. Estas ferrametas são fundamentais para fornecer mecanismos rápidos e acessíveis de rastreio e estratificação de amostras biológicas.

Neste contexto, este trabalho aborda (i) a optimização de um dispositivo protótipo existente responsável pela aquisição de dados de amostras biológicas e sua transmissão à cloud para posterior processamento e (ii) a concepção de uma estação de agitação orbital, controlada remotamente, para a homogeneização de amostras biológicas antes da aquisição de dados.

No estado actual do protótipo, taxa de transmissão de dados desde a aquisição até envio para a cloud não escala favorávelmente em relação ao número crescente de amostras biológicas que necessitam de ser analisadas. A abordagem adoptada para a primeira parte deste trabalho consistiu na análise de cada etapa da transmissão de dados, a fim de encontrar oportunidades de optimização.

Começando pela comunicação entre dispositivos utilizando o protocolo SPI, foi possível concluir, após cuidadosa investigação e avaliação da qualidade das formas de onda, que a escolha da frequência do relógio SPI estava abaixo dos níveis óptimos e podia ser optimizada em uma ordem de grandeza.

Relativamente à codificação, processamento e transmissão sem fios dos dados adquiridos, foram analisados todos os detalhes relacionados tanto com a codificação de dados como com a utilização do protocolo MQTT e Wi-Fi e foram identificados potenciais pontos de estrangulamento através de esaios reais e psterior análise estatística. Isto permitiu concluir que tanto o processo de codificação de dados como o protocolo Wi-Fi utilizado eram sub-óptimos e podiam ser optimizados em quase 30%. Já no que se refere ao protocolo MQTT, nenhuma melhoria possível foi identificada.

A segunda parte do trabalho consistiu no desenvolvimento de um agitador orbital para a homogeneização de amostras biológicas antes da aquisição de dados. Para este fim, após uma pesquisa de mercado adequada, foram escolhidos um motor brushless e respectivo controlador, e foi desenvolvidos o software necessário para controlar o motor bem como uma interface com o utilizador adequada a um dispositivo móvel para controlar a operação do agitador no tempo, direcção e velocidade de rotação. O projeto será testado num protótipo mecânico impresso em 3D.

Abstract

In the context of personalized medicine, progress has been made towards the integration of Photonics and Artificial Intelligence in the creation of a virtual library of disease biomarkers and biological profiles. These modern tools are fundamental to provide quick and accessible mechanisms for screening and stratification of biological samples.

In this context, this work addresses (i) the optimization of the existing prototype device responsible for the acquisition of data from biological samples and its transmission to the cloud for further processing and (ii) the design of a remote controlled orbital shaker station for the homogenization of biological samples prior to the data acquisition.

In the current state of the prototype, the data throughput from acquisition to transmission does not scale favorably in relation to the increasing number of biological samples needing to be analyzed. The approach taken for the first part of this work consisted in the analysis of each data transmission step in order to find throughput optimization opportunities.

Starting from the intra-device communication using the SPI protocol, it was possible to conclude, after careful waveform investigation and waveform quality assessment that the choice of the SPI clock frequency was below optimal levels and could be optimized by close to an order of magnitude.

Regarding the codification, processing and wireless transmission of data, details related to both data encoding and the MQTT and Wi-Fi protocol usage were studied and potential bottleneck points were identified through experiment and statistical analysis. This led to the understanding that both the data encoding scheme as well as the Wi-Fi protocol used were sub-optimal and could be optimized by almost 30%. As for the MQTT protocol, no possible improvements have been identified.

The second part of the work focused on the development of an orbital shaker for the homogenization of biological samples prior to data acquisition. For this purpose, a suitable brushless DC electric motor and controller were chosen after a proper market search, the required software to control the motor and a suitable user interface using a mobile device were developed to control the operation in time, direction and rotation speed. The design will be tested in a 3D printed mechanical mock-up.

Résumé

Dans le contexte de la médecine personnalisée, des progrès ont été réalisés vers l'intégration de la photonique et de l'intelligence artificielle dans la création d'une bibliothèque virtuelle de biomarqueurs de maladies et de profils biologiques. Ces outils sont fondamentaux pour fournir des mécanismes rapides et accessibles de dépistage et de stratification d'échantillons biologiques.

Dans ce contexte, ce travail porte sur (i) l'optimisation du dispositif prototype existant responsable de l'acquisition de données à partir d'échantillons biologiques et de leur transmission au cloud pour un traitement ultérieur et (ii) la conception d'une station d'agitation orbitale télécommandée pour l'homogénéisation des échantillons biologiques avant l'acquisition de données.

Dans l'état actuel du prototype, le débit de données entre l'acquisition et la transmission n'évolue pas favorablement par rapport au nombre croissant d'échantillons biologiques à analyser. L'approche adoptée pour la première partie de ce travail a consisté à analyser chaque étape de transmission des données afin de trouver des opportunités d'optimisation du débit.

En commençant par la communication intra-dispositif utilisant le protocole SPI, il a été possible de conclure, après une enquête minutieuse et une évaluation de la qualité des formes d'onde, que le choix de la fréquence d'horloge SPI était inférieur aux niveaux optimaux et pouvait être optimisé de près d'un ordre de grandeur.

En ce qui concerne la codification, le traitement et la transmission sans fil des données, les détails liés à la fois à la codification des données et à l'utilisation des protocoles MQTT et Wi-Fi ont été étudiés et les points d'étranglement potentiels ont été identifiés par l'expérimentation et l'analyse statistique. Cela a permis de comprendre que le schéma d'encodage des données ainsi que le protocole Wi-Fi utilisé étaient sous-optimaux et pouvaient être optimisés de près de 30%. Quant au protocole MQTT, aucune amélioration possible n'a été identifiée.

La deuxième partie du travail a porté sur le développement d'un agitateur orbital utilisé pour l'homogénéisation des échantillons biologiques avant l'acquisition des données. À cette fin, un moteur électrique sans balais et un contrôleur ont été choisis après une recherche de marché appropriée, le logiciel requis pour contrôler le moteur et une interface utilisateur utilisant un dispositif mobile ont été développés pour contrôler l'opération dans le temps, la direction et la vitesse de rotation. La conception sera testée dans une maquette mécanique imprimée en 3D.

Acknowledgments

To my parents, for always supporting my education.

To Professor João Paulo de Sousa, for all the knowledge and experience shared, his thoughtful input and unwavering support.

To Paulo Santos, for his commitment, support and availability.

Mário Cardoso

"What one fool can do, another can."

(Ancient Simian Proverb.)

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Motivation . . . 1
- 1.3 Goals . . . 2
- 1.4 Reading Guide . . . 2

2 Problem Characterization **3**
- 2.1 Problem Definition . . . 3
  - 2.1.1 Device Communication . . . 3
  - 2.1.2 Laboratory Shaker . . . 4
- 2.2 Proposed Solutions . . . 4
  - 2.2.1 Device Communication . . . 4
  - 2.2.2 Laboratory Shaker . . . 6

I Optimization of the IoT Communications Chain **7**

3 Literature Review **9**
- 3.1 Communication Protocols and Software . . . 9
  - 3.1.1 MQTT . . . 9
  - 3.1.2 HTTP . . . 11
- 3.2 ESP32 Wi-Fi Microcontroller Board . . . 12

4 Optimization of Internal Communications **15**
- 4.1 Current Implementation . . . 16
- 4.2 Test Methodology . . . 16
- 4.3 Conclusions . . . 18

5 Optimization of External Communications **19**
- 5.1 Data Fragmentation . . . 19
  - 5.1.1 Timer Validation . . . 20
  - 5.1.2 Experimental Setup . . . 20
- 5.2 Message Encoding . . . 23
  - 5.2.1 Experimental Setup . . . 24
- 5.3 MQTT QoS Level . . . 25
  - 5.3.1 Experimental Setup . . . 26
- 5.4 IEEE 802.11 Standard . . . 27
  - 5.4.1 Experimental Setup . . . 28
- 5.5 Conclusions . . . 29

II Development of a Laboratory Microplate Shaker **31**

6 Laboratory Orbital Shaker **33**
6.1 Context . . . 33
6.2 Market Research . . . 33
6.2.1 Overview . . . 33
6.2.2 Device Requirements . . . 36
6.3 Choice of Components . . . 36
6.3.1 Microcontroller . . . 36
6.3.2 Motor . . . 37
6.3.3 Motor Controller Board . . . 38
6.4 Interfaces . . . 39
6.4.1 Machine-to-Machine Interface . . . 39
6.4.2 Human-Machine Interface . . . 40
6.5 Control Approach . . . 42
6.6 Mechanical Assembly . . . 43
6.7 Results . . . 44

7 Conclusions **45**
7.1 Contributions . . . 45
7.2 Future Work . . . 46

A Base64 Decoding Script **47**

B Automation Script **49**

C Work Plan **51**

References **55**

List of Figures

1.1 Prototype device. 1

2.1 Device communication system concept. 4
2.2 96 well plastic microplate [1]. 5
2.3 Data transmission overview. 5

3.1 MQTT architecture. 10
3.2 HTTP architecture. 11
3.3 ESP32 WROOM-32 development board [2]. 13

4.1 SPI subsystem 15
4.2 Beginning of the SPI message at 9.375 MHz. 16
4.3 Partial capture of the SPI message at 9.375 MHz, with speed corrected output pin. 17
4.4 SPI SCK signal - slow GPIO speed (left), medium GPIO speed (right). 18

5.1 Transaction times of fragmented publication - 4 fragments per transaction. 21
5.2 Transaction times of non-fragmented publications - 1 fragment per transaction. . 21
5.3 Transaction times (in seconds) for 25 fragments of 16616 bytes. 22
5.4 Transaction times of Base64 encoded payload. 25
5.5 Transaction time (s) of each run of 100 transactions, fragmented and non-fragmented, for both QoS levels 1 and 2. 27
5.6 Transaction time of 100 transactions of four fragments of 16616 bytes. 28

6.1 Thermo Scientific Digital Plate Shaker [3]. 34
6.2 IKA MTS 2/4 digital microplate shaker [4]. 34
6.3 QInstruments BioShake 3000 elm [5]. 35
6.4 Espressif ESP32-S2-Saola-1 development board [6]. 37
6.5 Shaker device system concept. 37
6.6 42BLF01 brushless motor [7]. 38
6.7 Mikroe Brushless 7 Click control board [8]. 39
6.8 moserial, a serial terminal program. 41
6.9 The ESP RainMaker-based mobile remote interface. 42
6.10 Example controller speed curve [9]. 43
6.11 Two views of the orbital shaker prototype. 44

C.1 Gantt diagram for the first phase of the work plan (PDI). 53
C.2 Gantt diagram for the second phase of the work plan. 54

List of Tables

3.1 Comparison of the MQTT and HTTP protocols - adapted and abridged from [10]. 12
3.2 ESP32 Wi-Fi throughput as measured by Espressif 13

4.1 SPI pins choice for the LX7 MCU 17

5.1 Timer function validation using a oscilloscope. 20
5.2 Descriptive statistics of fragmented and non-fragmented publications. 21
5.3 Average median time (in seconds) to publish different numbers of data fragments. 22
5.4 Descriptive statistics for 25 fragments of 16616 bytes. 23
5.5 Descriptive statistics of fragmented publication of a Base64 encoded payload. . . 24
5.6 Fragmented and non-fragmented publications at QoS levels 1 and 2. 26
5.7 Descriptive statistics of fragmented publications using different IEEE 802.11 amendments. 29

6.1 Comparison overview of selected shaker devices characteristics. 35
6.2 Available serial commands and responses. 40
6.3 Common and distinguishing characteristics of the developed orbital shaker. . . . 44

List of Abbreviations and Symbols

ADC	Analog to Digital Converter
AI	Artificial Intelligence
BLE	Bluetooth Low Energy
DAC	Digital to Analog Converter
GPIO	General Purpose Input/Output
HAL	Hardware Abstraction Layer
HTTP	HyperText Transfer Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
M2M	Machine to Machine
MQTT	Message Queuing Telemetry Transport
MCU	Microcontroller Unit
PCB	Printed Circuit Board
PSRAM	Pseudo Static RAM
PWM	Pulse-Width Modulation
QoS	Quality of Service
RTOS	Real-Time Operating System
RFC	Request For Comments
SPI	Serial Peripheral Interface
SoC	System on a Chip
TCP	Transmission Control Protocol
TLS	Transport Layer Security
URI	Uniform Resource Identifier
UDP	User Datagram Protocol
UI	User Interface

Chapter 1

Introduction

1.1 Context

Research trends in healthcare-related developments reveal the need to develop tools that allow personalized diagnostics and therapeutics. Taking into account that both medicine and biotechnology are working towards micro and nanoscale methodologies, the need for intelligent microdevices arises. One of the most well-regarded candidates for these applications are optical fiber tools due to their flexibility, small size, and chemical inertness. This has contributed to the advent of the "Lab on Fiber" (LoF) devices [11].

The company behind the technology is a MedTech startup that intends to start a new era of personalized medicine, through the usage of AI and photonics for a virtual cloud-based library of disease biomarkers and biological profiles to facilitate quick and accessible mechanisms for both screening and stratification.

1.2 Motivation

Given the rapidly growing pace of the startup, it became important to ensure an increased sample processing capability by maximizing the automation level of the system.



Figure 1.1: Prototype device.

Figure 1.1 illustrates a currently developed prototype device, developed by the company, that allows data acquisition, sample analysis and upload of the results to the cloud. However, its testing capabilities are limited by the current device-cloud communication capacity that scales to an unreasonable amount of time for larger sample numbers. It is also important to increase the current throughput analysis capabilities, expanding them.

1.3 Goals

The main goal of the present dissertation is to design and test a solution for the automation of some processes of the integrated platform in order to increase its sample handling capabilities. In the first stage, the work will address the optimization of the device-cloud communication in order to obtain an increased data throughput. Additionally, an highly controllable and precise homogenization subsystem for samples in microplates will be developed so that the larger amount of samples waiting to be processed remain in appropriate condition.

1.4 Reading Guide

This document is organized into 7 chapters and 3 annexes:

Chapter 1 presents the motivations behind the project and introduces the context and objectives for the work.

Chapter 2 characterizes the two problems (optimization of communications and design of a laboratory shaker) and suggests possible ways forward to find adequate solutions.

Chapter 3 presents the literature review on the different areas of this work - MQTT and HTTP protocols and development boards based on the Espressif ESP32 SoC.

Chapter 4 presents the work done on the optimization of the internal communications (SPI communication between the two microcontrollers) of the prototype.

Chapter 5 presents the work done on the optimization of the external communications - message fragmentation, data encoding, MQTT Quality of Service and Wi-Fi link.

Chapter 6 presents the work done on the design of a laboratory shaker, addressing the market research, the choice of components, the development of a machine-to-machine interface and a human-machine interface and the motor control approach.

Finally, chapter 7 summarizes the contributions of this work and suggests possible future developments.

Annex A presents a simple, self-contained reference MQTT subscriber and message decoder script.

Annex B presents an example of a Python script illustrating the use of the machine-to-machine interface.

Annex C presents the work plan description and timeline.

Chapter 2

Problem Characterization

In this chapter, we present and define the proposed problem, and we introduce possible approaches to solving it based on preliminary discussions and brainstorming sessions.

2.1 Problem Definition

The concept behind the "Lab on fiber technology" is to apply Artificial Intelligence techniques to the back-scattered signal arising from a trapped cell by a micro-lensed optical fiber. The gathered data allows the creation of a cloud-based library of disease-related biomarkers and biological profiles, enabling rapid and accessible triage and stratification mechanisms [11]. This concept has quite large device-cloud communication needs and the optimization of the communication process is precisely the core of this work.

2.1.1 Device Communication

The platform needs to transmit the outputs resulting from the biological sample analysis to the cloud for further processing. The main problem to deal with is the slow device-cloud transmission speed available in the current integrated prototype. While it is acceptable for a low number of samples to test, it scales to excessive wait times for increased test counts.

The information flow of the current prototype device starts with the data acquisition by a dedicated ARM microcontroller-driven platform. The data is then passed to a Wi-Fi-enabled ESP32 microcontroller-based module that connects to a local Wi-Fi network, behaving as a client device. The acquired data is mapped to a JSON file following a set schema and then serialized for transmission to the message broker. These communications rely on the MQTT application protocol on the ESP32-side through its MQTT implementation and on the broker-side running the Eclipse Mosquitto[1] message broker software. The information received by the broker is then relayed to a cloud-based IT solution for further processing and analysis.

The purpose of this work is to test the limits of the current communication implementation through the analysis of problematic sections in the chain of communication encompassing the

[1] Eclipse Mosquitto, open-source MQTT message broker - https://mosquitto.org/

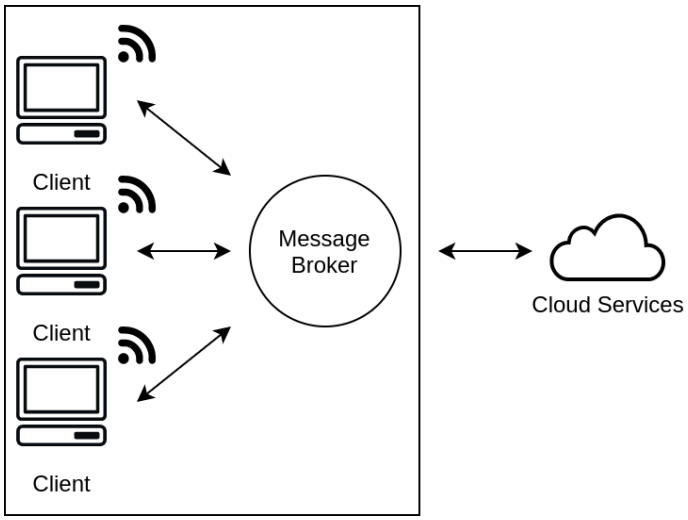


Figure 2.1: Device communication system concept.

boxed section in figure 2.1. These potential bottlenecks may exist in the different stages of data parsing, encoding, decoding, transmission between endpoints or possibly in the sub-optimal configuration of support software and libraries. The outcome of this work will be a set of recommendations for the optimization of the communications chain.

2.1.2 Laboratory Shaker

Once the communication bottlenecks are identified and eliminated the goal of increasing the capacity of biological analysis can be addressed. However, increasing the number of biological samples that can be processed in a given time interval triggers another problem: the need for homogenizing the samples waiting to be processed.

Samples are usually contained in microplates. A microplate is a plate consisting of an array of wells capable of holding a small bio-sample for experimentation to take place. Microplates are typically available in different sizes such as 96, 384 and 1536 wells per plate [12]. For this application, a 96 well microplate will be considered, such as the one in figure 2.2.

To solve this homogenization problem, an integrated microplate shaker will be developed. The microplate shaker device should feature an orbital motion on two axes with an orbit in the millimeter range, and be able to have configurable mixing time, direction and speed. The equipment needs to integrate well into an automated laboratory workflow, as such, it should be able to be remotely commanded through a serial interface, in addition to a human interface.

2.2 Proposed Solutions

2.2.1 Device Communication

As stated in the previous section, in order to make the device communication throughput compatible with an increasing number of samples to test, one of the issues to tackle is the amount of time



Figure 2.2: 96 well plastic microplate [1].

it takes to relay the data acquired at the input source all the way to the cloud-based infrastructure responsible for its processing and analysis.

In order to better identify and understand all the potential improvement opportunities, each stage of data acquisition, transmission as well as data processing and configuration of the prototype device was divided into sections as shown in figure 2.3.

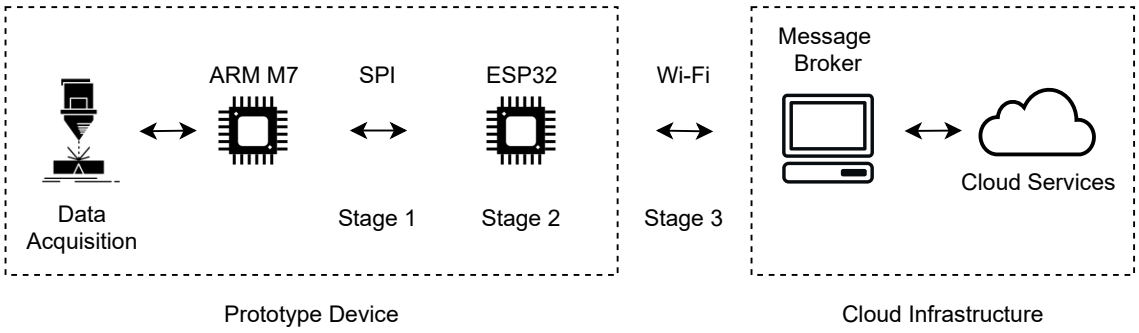


Figure 2.3: Data transmission overview.

The first stage consists in the transmission of the acquired input data between the microcontrollers present in the client device. An ARM microcontroller is responsible for the data acquisition and an ESP32 module is responsible for the communication with the broker. These two microcontrollers communicate through a Serial Peripheral Interface (SPI). Even though not many performance improvement opportunities are expected, both sides of the communication will be checked for possible configuration improvements and adequate clock frequency.

At the second stage, the data present at the second microcontroller is mapped according to a given JSON schema and then serialized in order to be published. Several improvement opportunities arise at this stage, such as ensuring an efficient mapping to the JSON object, ensuring the resulting data is properly encoded for transmission, searching for improvement opportunities in the payload, ensuring the payload is efficiently transmitted and that optimal settings are set for the message to be sent.

During the third stage, the client device engages with the MQTT message broker through a Wi-Fi network connection. Aspects such as the Wi-Fi standard choice of the client device and the type of antenna in use[2] are relevant factors for optimizing the transmission speed and reliability.

The inclusion of a wired network connection could be considered in order to improve the previously referenced metrics. Although this approach would drastically improve the data communication throughput, it would also reduce the portability of the equipment and for that reason, it will not be further considered in this work.

It is expected that by a thorough analysis of the system, all the critical points in the communication chain will be detected and relevant optimization recommendations can be formulated.

2.2.2 Laboratory Shaker

In order to solve the previously mentioned sample homogenization problem, an integrated orbital microplate shaker needs to be developed.

Like in all modern laboratory shakers, the control should be digital and based around a microcontroller unit. The microcontroller will work as a central hub, bridging together user input, logic processing and motor drive control.

A power management block should provide power levels adequate for the microcontroller unit, the motor drive and eventually other peripherals. Particular care should be taken with motor drive power delivery as relatively high current needs are expected.

Since the device is geared towards integration and automation, most of the user interaction will happen through serial communication, set up and dealt with by the microcontroller unit. Additionally, a simple local user interface should provide a straightforward interaction point for simple operations and could also aid during the development process. Both interfaces should be able to work concurrently.

The proposed microplate shaker build should provide an adequate solution to the sample homogenization problem while integrating with existing laboratory equipment.

[2] Only some ESP32 modules provide an external antenna connector [13].

Part I

Optimization of the IoT Communications Chain

Chapter 3

Literature Review

This chapter provides an overview of the literature review on the different areas of this work - MQTT and HTTP protocols and development boards based on the Espressif ESP32 SoC. The market research carried out to address the second part of this work is not reported here but in the correspondent chapter.

3.1 Communication Protocols and Software

This section describes and analyses protocols and software already in use at the current stage of the prototype and further explorers new potentially viable alternatives. It is restricted to the two protocols supported by the current cloud infrastructure as using other protocols would imply major changes at the cloud level that are out of the scope of the current work proposal.

3.1.1 MQTT

The Message Queuing Telemetry Transport (MQTT) is a messaging protocol originally developed by Stanford-Clark of IBM and Arlen Nipper of Eurotech and first released in 1999 [10]. The protocol was first standardized in 2013 by the Organization for the Advancement of Structured Information Standards (OASIS) and later in 2016 by the International Organization for Standardization (ISO) [14][15]. MQTT is a binary protocol and its message payload size is limited to 256 MB. It has a fixed header length of only two bytes resulting in reduced overhead. As such, its lightweight and simple design make it suitable for use in low resource devices and constrained networks such as those often present in Machine to Machine (M2M) and Internet of Things (IoT) contexts [10][14]. MQTT usually runs on top of TCP/IP protocols but it is capable of working over other network protocols that offer ordered, lossless and bidirectional communications. By itself, MQTT is not secure and relies on layering on top of the Transport Layer Security (TLS) protocol to ensure security [14].

In the MQTT protocol, the transport protocol provides the means to a network connection that clients and servers can connect to and send streams of bytes bidirectionally. As depicted in figure

3.1, the protocol follows a publish-subscribe model where the publisher posts messages to a message broker, and the subscriber receives from the broker the application messages it is interested in. This message can also be retained for newer subscribers to receive. These communications happen from the publisher through the means of application messages that contain the payload data, quality of service information, a number of properties and a topic name [14][10].

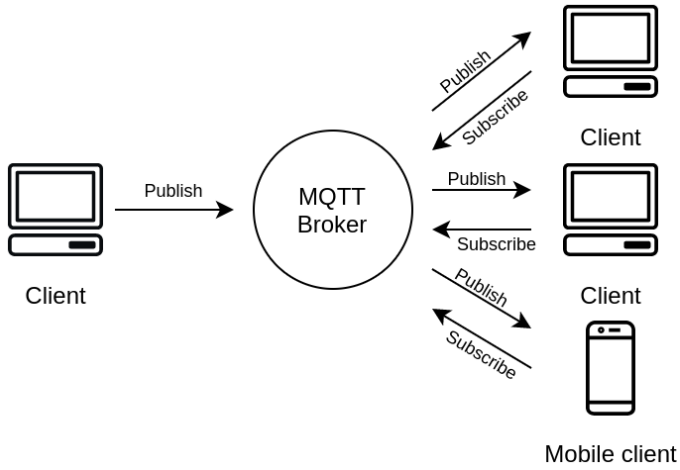


Figure 3.1: MQTT architecture.

The client is a device or a program that is able to publish messages that other clients can access and request to subscribe to or unsubscribe from a given topic. Clients can subscribe to multiple topics and receive all the messages published to each of them.

The server is a device or a program that serves as a broker between the clients that publish a message and clients that subscribe to a topic. It deals with subscribe and unsubscribe requests from clients and accepts and forwards messages to clients that match a given subscription. This pattern decouples the clients from each other as the publishers are kept unaware of the existence of the subscribers.

Each application message associates with a given topic defined as an address. This address can have hierarchical levels that can be subscribed to as a whole through the usage of the + and # wildcards. Receivers can use the + wildcard for a single hierarchy level and # multi-level wildcard representing the parent and all the child levels.

The way application messages are delivered depends on the Quality of Service (QoS) level defined. These delivery considerations apply to the delivery of a message from one single sender to a single receiver. In case multiple clients should receive a message each client is treated independently.

At QoS 0, the lowest level, each message is delivered at most once and is subject to the capabilities of the underlying transport protocol. That means there is no delivery confirmation and the sender does not retry sending.

QoS 1 ensures that the message is sent at least once. The sender considers the packet as "unacknowledged" until receiving confirmation from the receiver.

QoS 2, the highest level, ensures the message delivery occurs exactly once, at the cost of additional overhead. The receiver of a message with this QoS level acknowledges receipt with a two-step acknowledgment process.

Since the client can send a message at any QoS level and clients can subscribe up to a given QoS level, the QoS level used to deliver a message outbound to the client could be different from the inbound message [14].

3.1.2 HTTP

The HyperText Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems, first developed by Tim Berners-Lee at CERN in 1989 [16]. It was first specified as a standard in 1997, referred to as HTTP/1.1, by a joint effort by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C). Further revisions and related Request for Comments (RFC) were later released by the IETF. HTTP usually runs on top of TCP/IP protocols but it is capable of working over other network protocols that provide a means of reliable transport [17].

By itself, HTTP is not secure and relies on layering on top of the Transport Layer Security (TLS) protocol to ensure security. This results in HTTP over TLS, known as HTTP/TLS. The usage of a different server port can be used to distinguish between secured traffic and insecure traffic. However, through an upgrade mechanism, the same server port used for HTTP can be used for HTTP/TLS. HTTP/TLS is distinguishable from HTTP URIs by the usage of the 'https' protocol identifier instead of the 'http' protocol identifier [18].

As shown in figure 3.2, HTTP follows the client-server model, where a client device sends a request to a server which then handles the response.

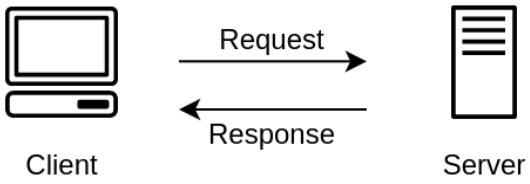


Figure 3.2: HTTP architecture.

HTTP uses Uniform Resource Identifiers (URI) that identify the location of resources, which can be absolute or relative to another base URI.

The header size is not specified or restricted by the standard, and it is up to each server implementation to decide up to what size they are able to receive. Similarly, the maximum content length is also not specified and is usually limited by practical and safety considerations [17].

Other protocols, more adequate to an IoT scenario than HTTP, could have been considered in this analysis. Unfortunately, given the required changes to other parts of the whole system they would imply, namely at the level of the cloud provider, would have made this approach out of the context of the work proposal.

Table 3.1: Comparison of the MQTT and HTTP protocols - adapted and abridged from [10].

Criteria	MQTT	HTTP
1. Year	1999	1997
2. Architecture	Client/Broker	Client/Server
3. Abstraction	Publish/Subscribe	Request/Response
4. Header Size	2 Byte	Undefined
5. Message Size	Small and Undefined (up to 256 MB maximum size)	Large and Undefined (depends on the web server or the programming technology)
6. Semantics/Methods	Connect, Disconnect, Publish, Subscribe, Close	Get, Post, Head, Put, Patch, Options, Connect, Delete
7. Cache and Proxy Support	Partial	Yes
8. Quality of Service (QoS)/Reliability	QoS 0 - At most once QoS 1 - At least once QoS 2 - Exactly once	Limited (via Transport Protocol - TCP)
9. Standards	OASIS, ISO	IETF
10. Transport Protocol	Typically TCP	Typically TCP
11. Security	TLS/SSL	TLS/SSL
12. Default Port	1883/8883 (TLS)	80/443 (TLS/SSL)
13. Encoding Format	Binary	Text
14. Licensing Model	Open Standard	Open Standard
15. Organisational Support	IBM, Facebook, Eurotech, Cisco, Red Hat, Amazon Web Services	Global Web Protocol Standard

3.2 ESP32 Wi-Fi Microcontroller Board

The ESP32 is a popular, low-power, low-cost system on a chip (SoC) featuring integrated Wi-Fi and Bluetooth connectivity. It can contain either a single-core or dual-core Xtensa 32-bit LX6 microprocessor with an adjustable frequency from 80 MHz to 240 MHz, alongside an ultra-low-power co-processor [19]. Espressif Systems also makes development boards based on the module, such as the one in figure 3.3.

Besides having 520 KB of internal SRAM, Espressif ESP32 modules such as the ESP32-WROVER can contain either a PCB antenna or an IPEX connected one, 4 MB of external SPI flash and 8 MB SPI pseudo-static RAM (PSRAM) [20]. The ESP32 contains peripheral interfaces for a 12-bit ADC, two 8-bit DAC, four SPI, two I2S interfaces, two I2C interfaces, three UART interfaces, one IR remote controller interface, CAN, 34 programmable GPIO, among others.

The device features security considerations such as flash encryption, secure boot to prevent running unverified code as well as cryptographic hardware acceleration for AES, SHA-2, RSA, ECC and random number generation.

Its wireless communication capabilities through Wi-Fi follow the IEEE 802.11 b/g/n protocols, where the 802.11n protocol sets a throughput of up to 150 Mbps on the 2.4 GHz band. Additionally, it features an integrated Bluetooth module compliant with Bluetooth version 4.2 as well as

the Bluetooth Low Energy (BLE) protocol [19].

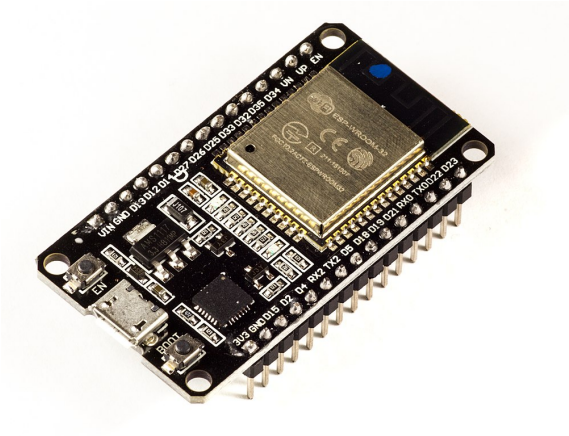


Figure 3.3: ESP32 WROOM-32 development board [2].

Espressif provides official ESP32 libraries for some application protocols such as MQTT, HTTP, HTTPS, TLS, ASIO and Modbus [21].

The manufacturer provides an overview of their best case, laboratory obtained Wi-Fi throughput of the ESP32 under different scenarios, both in open air and inside an electromagnetic shielding box, using an ESP-IDF SDK version as set by the 05838641 git commit id [22]. Those results are summarized in table 3.2.

Table 3.2: ESP32 Wi-Fi throughput as measured by Espressif

Type/Throughput	Air In Lab	Shield-box
Raw 802.11 Packet RX	N/A	130 Mb/s
Raw 802.11 Packet TX	N/A	130 Mb/s
UDP RX	30 Mb/s	90 Mb/s
UDP TX	30 Mb/s	60 Mb/s
TCP RX	20 Mb/s	50 Mb/s
TCP TX	20 Mb/s	50 Mb/s

These results can be seen as an optimistic upper bound throughput of the most common transport layer protocols. As such, transmissions occurring through application layer protocols layered on top of either of them are extrinsically limited to a relatively slower throughput due to their own overhead.

Considering an average input size of 120 kB per sample and a target transmission time of 1 second, a throughput of approximately 1 Mb/s is needed to match this requirement. Comparing the desired throughput with the hardware's theoretical TCP transmission capabilities, the theoretical throughput is about 20 times the desired one. As such, it is possible to conclude that the ESP32 can be considered as a viable option to explore for this work.

Chapter 4

Optimization of Internal Communications

In the current prototype, SPI communication is used for the communication of commands and data between the two microcontroller-driven units. One of them is a 32-bit performance-oriented ARM Cortex-M-based MCU and the other is an Espressif ESP32-S2 module, containing an Xtensa 32-bit LX7 microprocessor, connected as illustrated in figure 4.1. This chapter presents the work done on the optimization of the SPI communication between the two microcontrollers of the prototype.

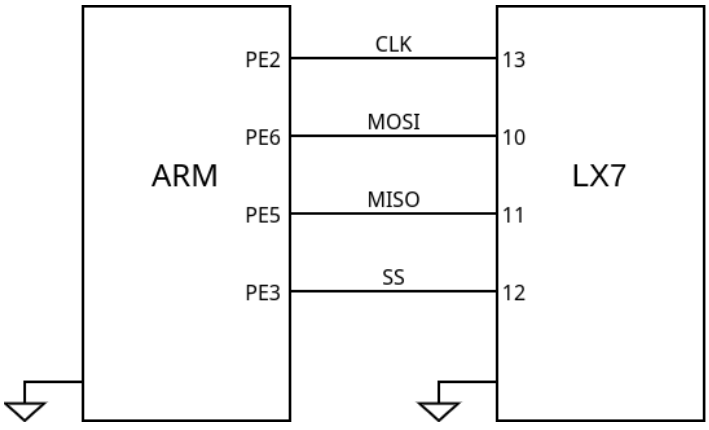


Figure 4.1: SPI subsystem

The main questions that arise pertaining to the current setup are whether the currently implemented SPI communication can be improved and whether the SPI communications are a limiting factor of the overall communication chain. In this chapter, we will answer the first question and will leave the latter for future considerations after additional avenues for optimizing the overall communication throughput are explored.

4.1 Current Implementation

In the SPI bus, the ARM MCU behaves as the SPI master device, outputting the Serial Clock (SCLK), Master Output Slave Input (MOSI) and Slave Select (SS) signals. The LX7 MCU is programmed to behave as the slave device on the bus, taking the aforementioned SCLK, MOSI and SS signals as inputs and outputting the Master Input Slave Output (MISO) signal. Despite the flow of data being bi-directional, the bulk of the data is transferred from the ARM to the LX7 MCU.

In its current implementation, the bus master defines its working mode as full-duplex, has a data size of 8 bits and that the data is shifted out with the most significant bit first. For the clock configuration, it sets the Clock Polarity (CPOL) so the signal idles high (its leading edge is the falling edge) and sets the Clock Phase (CPHA) to the second edge, meaning that the data is changed in the SCLK signal leading edge, and sampled in its trailing edge. Finally, the bit rate is set to 750 kb/s. The ESP32-S2 software slave driver is configured accordingly to the master device settings.

4.2 Test Methodology

With the baseline configuration established, the next step consisted of the development of a test program to replicate and evaluate the results of the SPI message transactions between the two devices. First, each device loaded a predetermined 1024 bit message in its transmission buffer, consisting of 128 AAh bytes on the master device and 128 55h bytes on the slave device. Then, on user input, a transaction takes place where each device shifts out its transmit buffer into the other device receive buffer. Finally, each device internally verifies the integrity of the received data and communicates the results through a UART peripheral. Additionally, the master indicates either success or failure using status LED. An oscilloscope capture of the beginning of the SPI transaction at 9.375 MHz, showing good message integrity, is shown in figure 4.2.

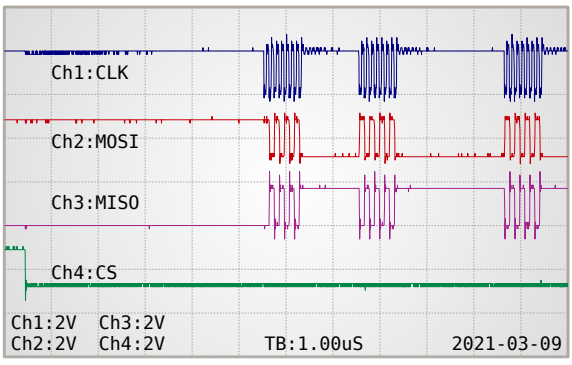


Figure 4.2: Beginning of the SPI message at 9.375 MHz.

Considering the ESP32-S2 slave driver frequency limit of 10 MHz, the maximum SCK frequency chosen for the SPI master was limited to 9.375 MHz. This frequency corresponds to

the largest possible value common to both MCU. Compared to the original 750kHz this already represents more than a 10× increase in the SPI bit rate.

In addition to the verification of message integrity, the signal quality also needed to be verified while both MCU communicate connected to the device PCB. Several tests were made using the same SPI settings as before with the exception of increasing the SCK frequency.

It is to be noted that ESP32-S2 slave driver maximum frequency is also limited by the inherent output delay on its MISO pin. In order for the MISO signal to be stable on the sampling edge, its output delay should not exceed half of the clock signal period. As per the documentation, if the SPI pins are routed through the GPIO matrix, they have an output delay of 68.75 ns and a frequency limit of 7.2 MHz. Otherwise if routed directly, the output delay is 43.75 ns and its maximum frequency of 11.4 MHz, not limiting the overall maximum SPI frequency [23].

In this case, the original choice of pins for the SPI subsystem, represented in figure 4.1, is sub-optimal, as it forces the GPIO matrix to be used and introduces an additional delay in the signal path.

Table 4.1: SPI pins choice for the LX7 MCU

	Current choice	Optimal choice
SCK	13	12
MOSI	10	11
MISO	11	13
SS	12	10

In order to change the LX7 SPI pins to the optimal choice as shown in table 4.1 it would require a redesign of the PCB, which falls outside the scope of this work but can be considered a suggestion for future improvement.

Upon closer inspection, some signals showed signs of voltage overshoot. This effect could be greatly minimized on the SPI master side by fine-tuning the relevant pin speed setting according to the frequency of the signal to be outputted. Faster GPIO frequencies provided faster slew rates but caused voltage overshoot. This suited higher frequency signals by better shaping their rising and falling edges.

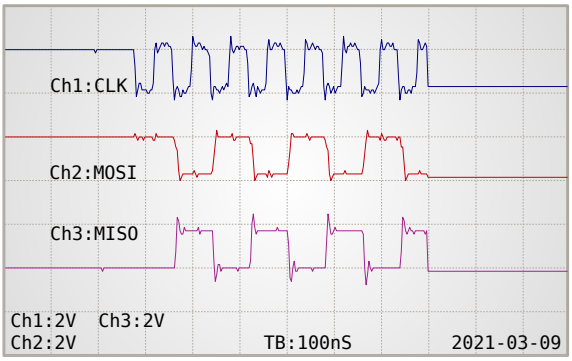


Figure 4.3: Partial capture of the SPI message at 9.375 MHz, with speed corrected output pin.

Slower GPIO speed settings provided a reduced voltage overshoot at the expense of slightly slower rise and fall times. A partial capture of an example message with the corrected output speed is shown in figure 4.3. Additionally, a close-up of the effects of a slow-speed and medium-speed GPIO can be seen in figure 4.4.

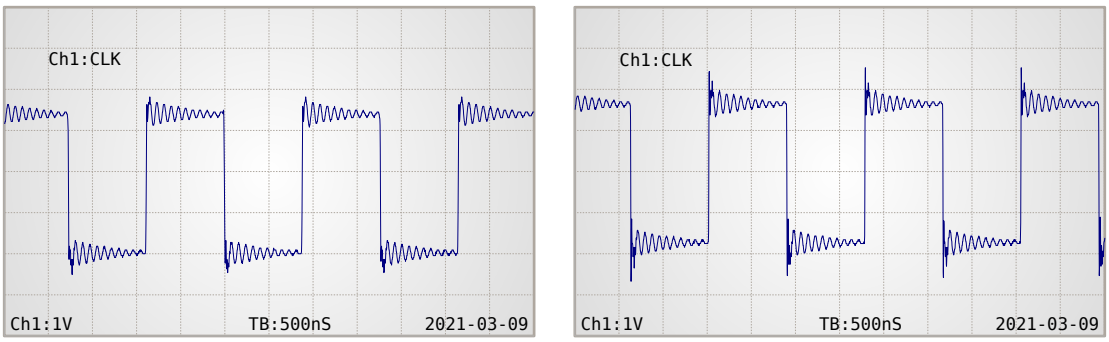


Figure 4.4: SPI SCK signal - slow GPIO speed (left), medium GPIO speed (right).

In both cases, the signal contains visible ringing on its high and low levels. These unwanted oscillations can be attributed to the presence of parasitic capacitances and inductances in the PCB circuit. Despite the development of a new PCB design being out of scope for the current work, this occurrence could be minimized with the usage of a small series resistor at the output of the signals in order to dampen the output of the parasitic oscillator circuit.

4.3 Conclusions

In conclusion, the answer to whether the SPI communication can be optimized is yes, as the SCLK can be increased 12.5 times, resulting in an improved bit rate of 9.375 Mb/s from the original 750 kb/s, and the answer to whether or not the optimization will be needed can not be stated yet; it will depend on the overall communication path that still has to be studied in detail. If the GPIO matrix of the LX7 MCU can not be avoided, the additional delay introduced in the signal path will force an additional decrease of the clock frequency from 9.375 MHz to 7.2 MHz. Even in this case, the improvement in speed will be significant: from 750 kHz to 7.2 MHz corresponds to an almost ten times increase in throughput.

Chapter 5

Optimization of External Communications

In this chapter, we analyze the communication between the ESP32-S2 client device and an MQTT server running the Mosquitto broker software.

Having a focus on performance optimization opportunities, our goal is to find the answer to the following questions, listed from the top to the bottom of the communication stack:

1. Does publishing fragmented data results in a considerable performance overhead compared to a single publish operation?
2. How is the acquisition data encoded? Is it the most appropriate method?
3. Which MQTT QoS level is more suitable for message publishing? Is there a significant performance difference between them?
4. Which IEEE 802.11 protocol is being used? Is it the optimal choice?

The second question, while not directly related with the communication process itself, is closely related to the moments before and after and may have a considerable impact in the system throughput.

5.1 Data Fragmentation

As a result of former limitations in the cloud infrastructure, the acquired data is currently published by the ESP32-S2 in several fragments to the MQTT broker. As those limitations have meanwhile been lifted, we decided to analyze the impact of data fragmentation in the MQTT communication throughput, to decide whether or not the fragmentation should be kept.

It is to be noted that despite the communication involving the ESP32-S2 being bidirectional, the bulk of data is published from the ESP32-S2 client to the MQTT server; that is the reason why this study concentrates on the publisher to broker communication.

In the current setup, the client and server devices are configured to use the MQTT standard version 3.1.1. All messages are published at QoS level 1, meaning it is guaranteed the message is delivered at least once to the receiver; in other words, the publisher keeps retrying until it receives a PUBACK packet for a given message identifier. The messages consist of a JSON string containing a header with relevant metadata, followed by the payload itself, the acquisition data.

The experiment consisted in developing a test program capable of sending a large number of messages, with different fragmentation characteristics and outputting meaningful data for analysis. This program allows to easily set up an arbitrary number of transactions, where a transaction consists of publishing sample acquisition data in any given number of fragments, and then waiting to receive an acknowledgment packet for each message published. The program would then output a timestamp of when the first publish command was sent and a timestamp of when the last acknowledgment packet was received for each transaction in a CSV formatted string for further processing and statistical analysis.

5.1.1 Timer Validation

A fundamental aspect of this study is to guarantee the correctness of the time-stamping mechanism. The timestamps were obtained using the *esp_timer_get_time()* function made available through the ESP-IDF SDK. This function makes use of a high-resolution timer to obtain the time passed since start-up with microsecond resolution. In order to be sure that the result is suitable for measuring the transaction times, another test program was developed to help validate the timer output against an oscilloscope measurement. In order for a comparison to be made, a GPIO pin was toggled immediately after each timer function call. The switching output time was measured with an oscilloscope and the timer output was reported via the serial port. The results are summarized in table 5.1.

Table 5.1: Timer function validation using a oscilloscope.

Timer (µs)	Oscilloscope (µs)	Percent Error
10	7.12	28.80%
21	20.96	0.191%
9422	9430	0.085%

There is a drastic reduction in the experimental error for times above 20 µs. As the transaction times are expected to be in the order of tenths of milliseconds, the percent error values confirm the output timestamp of *esp_timer_get_time()* to be adequate to measure the desired time intervals.

5.1.2 Experimental Setup

The first batch of tests consisted of the transmission of 100 transactions, each containing an acquisition message split in four fragments of 16616 bytes each. This was followed by the transmission of a second set of 100 transactions, this time each containing only a single message of 65768 bytes, corresponding to the same payload. This sequence of tests was repeated three times and

each transaction time, from publishing to receiving the acknowledgment from the broker, was recorded and plotted.

Figure 5.1 shows the results. Each point corresponds to a transaction, and the vertical axis denotes the time, in seconds, it took to complete, from the first message published to the last acknowledgment received.

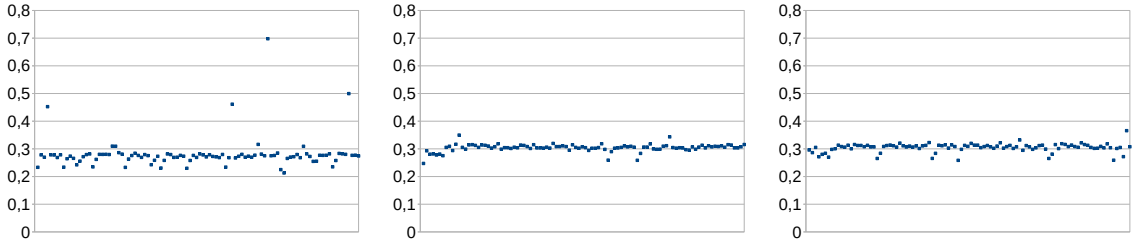


Figure 5.1: Transaction times of fragmented publication - 4 fragments per transaction.

Similarly, figure 5.2 shows three runs of one hundred transactions, containing a single fragment of 65768 bytes each, corresponding to the same payload sent previously.

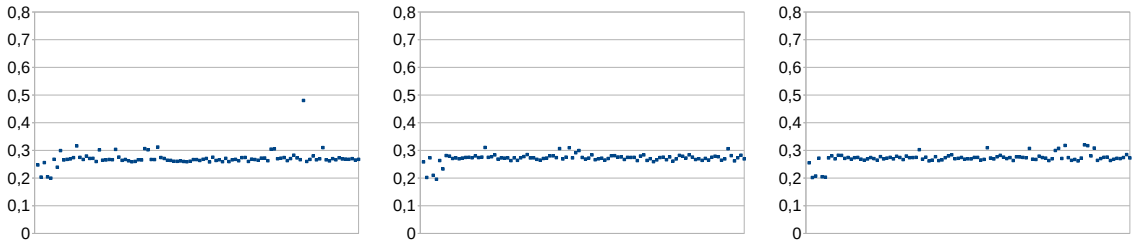


Figure 5.2: Transaction times of non-fragmented publications - 1 fragment per transaction.

These graphics are useful to give an overview of the results obtained but to be able to draw useful conclusions we need some simple descriptive statistics. Table 5.2 presents the mean and the median publication time for the six illustrated experiments.

Run	Mean (s)	Median (s)
1	0.280	0.275
2	0.304	0.305
3	0.305	0.308

(a) Fragmented publication.

Run	Mean (s)	Median (s)
1	0.271	0.267
2	0.272	0.273
3	0.273	0.273

(b) Non-fragmented publication.

Table 5.2: Descriptive statistics of fragmented and non-fragmented publications.

Taking these results into consideration, we can conclude that, on average, the transaction time of fragmented publication shows a 9% increase in relation to non-fragmented publication. This increase can be explained by the fact that the fragmentation causes an immediate overhead penalty as dividing the payload into several pieces implies that the message header needs to be repeated in every message containing the partial payload. In this case, for the same payload, the fragmented approach took an extra 696 bytes, which corresponds to a 1% increase over publishing a single

fragment per transaction. The remaining 8% difference is due to protocol overhead (increased function calls, increased number of acknowledgment packets, etc.).

To better assess the evolution of the transaction times with respect to the amount of data sent, the previous tests were repeated with a different number of fragments. The transmission time for 4, 8, 15 and 25 fragments saw a fairly linear increase for non-fragmented publication and a slightly polynomial increase for fragmented publication, allowing to easily extrapolate for other numbers of fragments.

Table 5.3: Average median time (in seconds) to publish different numbers of data fragments.

Data Fragments	4	8	15	25
Fragmented	0.25	0.44	0.78	1.34
Non-fragmented	0.22	0.44	0.82	-

For higher payloads the non-fragmented approach is even less attractive as the memory management mechanisms of the libraries kick in to address the increased memory needs, introducing additional time penalties. The break-even point is at 8 packets of 16616 bytes corresponding to a total payload of around 128 kBytes.

A special case is the test with 25 fragments - it is the most similar to the working conditions of the prototype and for that reason, we present here more details. The testing consisted of sending twenty-five 16616 bytes data fragments per transaction as well as single fragment transactions containing the same payload. Once again, due to header overhead, the fragmented messages contain a total of 415400 bytes, a 1.36% increase over the single message size of 409832 bytes.

In a similar way, each test run was repeated three times and each transaction from publishing to receiving the acknowledgment from the broker was timed, plotted and analyzed. Detailed results are presented in figure 5.3 and table 5.4.

Each point corresponds to a transaction, and the vertical axis denotes the time in seconds it took to complete, from the first message published to the last acknowledge.

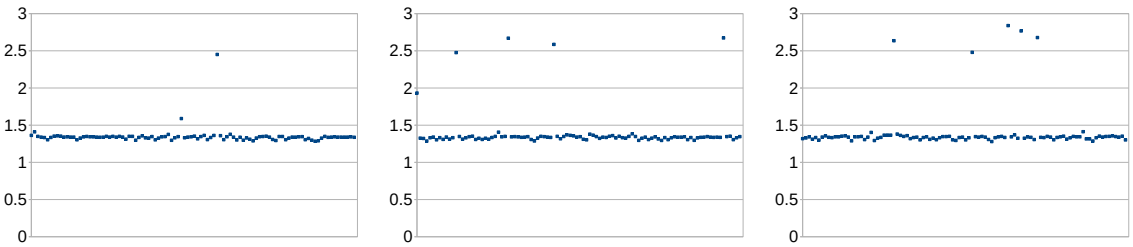


Figure 5.3: Transaction times (in seconds) for 25 fragments of 16616 bytes.

Table 5.4 summarizes some descriptive statistical measures of each of the three previous runs of one hundred transactions each.

The next step would be publishing the same payload as before, but this time in a single message. This single message would take 409832 bytes, which is larger than the ESP32-S2 SRAM of 320 kB. Attempts were made to place initialized variables into the external PSRAM but that

Table 5.4: Descriptive statistics for 25 fragments of 16616 bytes.

Run	Mean (s)	Median (s)
1	1.35	1.34
2	1.39	1.34
3	1.40	1.34

operation is not currently supported by the ESP-IDF SDK. For this reason, it was not possible to test the non-fragmented publication. However, a simple linear extrapolation using the results of table 5.3 gives an estimated time of 1.37 s.

Taking into consideration all the values obtained, it is possible to conclude that the time grows linearly with the size of the fragments and the fragmentation of messages incurs in a less than 10% time penalty. It is now possible to answer the first question: fragmented publishing does not result in a significant increase in overhead and is a good trade-off between time efficiency and memory footprint.

5.2 Message Encoding

The second question results from the need to adequately encode the acquisition data for transmission. For transmission purposes, the message payload needs to be encoded as plain text. Since the data is stored as a set of eight-bit binary elements, there is a need for a suitable mapping strategy for ASCII conversion. ASCII defines 128 characters, but since not all of them are printable, the selection pool becomes smaller.

The currently implemented strategy to deal with the message encoding directly translates the payload hexadecimal representation to a string of ASCII digits from ’0’ to ’9’ and characters from ’A’ to ’F’, also known as Base16 encoding. While this solves the binary-to-text conversion problem using only printable characters, each nibble is now represented by an eight-bit byte. This is potentially problematic as it effectively doubles the payload size prior to transmitting. In fact, this 50% data efficiency leads to an 8 kB payload to be encoded to a 16 kB message prior to transmitting. While it is reasonably expected that a conversion of this type to incur in an overhead penalty, a 100% payload size increase stands out as a possible improvement opportunity.

A viable and commonly used alternative is Base64 encoding. Like Base16, it does binary-to-text encoding from a series of 8-bit binary elements to an ASCII string consisting of printable characters. Compared to Base16, the available index table is vastly expanded to include characters from ’A’ to ’Z’, ’a’ to ’z’, digits from ’0’ to ’9’, symbols ’+’ and ’/’ plus ’=’ for padding. The input byte array is subdivided into groups of 6 bits, providing the right amount of combinations ($2^6 = 64$) to fit the index table, which are then attributed their Base64 character value. As a result, Base64 encoding provides a 75% data efficiency, compared to the 50% data efficiency of Base16. This leads to an 8 kB payload being encoded to a 10.64 kB message instead of 16 kB message before being transmitted.

Considering the improved data efficiency, it is expected the payload size to decrease 33% when going from Base16 to Base64 and that the transmission time to also decrease by about the same amount.

A direct consequence of changing the payload encoding scheme is that the data received downstream by the relevant subscribing devices needs to be decoded differently in order to extract any meaningful content. As a measure to aid the development of this new parser, listing A.1 in appendix A shows a simple, self-contained reference implementation consisting of an MQTT client working as a subscriber, a JSON string parser and a Base64 payload decoder that was developed using the Ruby programming language and provided to the company.

This example makes use of the MQTT gem as well as the JSON and Base64 standard libraries. It starts by establishing a connection to a given address and port, which then yields a connection handle. The handle is used to subscribe to a topic after which the subscribe method yields both the topic name and message content for each message received. Finally, the message data is converted to a JSON object and the payload is decoded and stored.

5.2.1 Experimental Setup

Having the current algorithm identified and prospective alternatives considered, a test program was developed to better understand the actual implications of the new encoding scheme to the data throughput. Prior to publishing, a sample of raw acquisition data is encoded in Base64 using the *mbedtls_base64_encode()* function from the Mbed TLS library included in the ESP-IDF SDK and the final JSON string is assembled by the ESP32-S2. Afterward, the program sets up a number of transactions consisting of publishing a number of data fragments each containing a Base64 encoded payload. Like in the previous tests, each transaction is timed by time-stamping the first message published and the last confirmation packet received using the *esp_timer_get_time()* function.

Each test batch comprised of the transmission of one hundred transactions containing an acquisition message split in four fragments of 11117 bytes each and also in the transmission of a bigger message, this time split in 25 fragments of 11117 bytes each.

Each batch of tests mentioned before was run three times each and a representative run of each was chosen and displayed in figure 5.4. Each point corresponds to a transaction, and the vertical axis denotes the time in seconds it took to complete, from the first message published to the last acknowledgement packet received. Table 5.5 shows the mean and median transaction times for each run of 4 and 25 fragments.

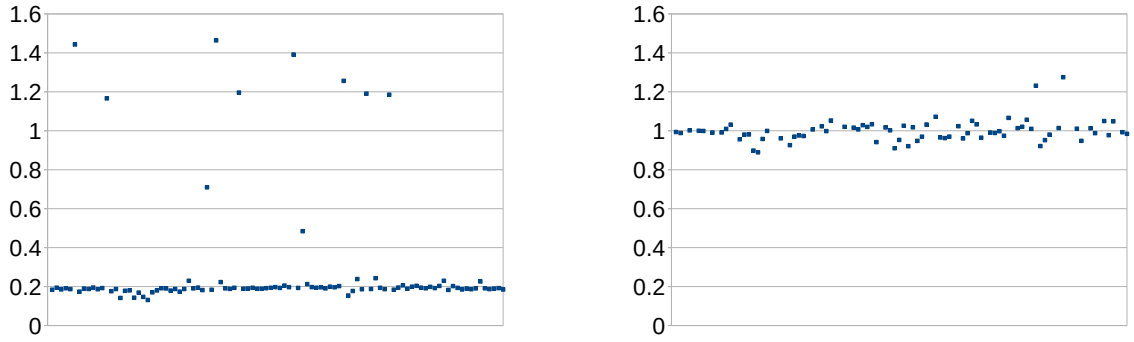
Run	Mean (s)	Median (s)
1	0.31	0.19
2	0.29	0.19
3	0.31	0.19

(a) Four fragment publication.

Run	Mean (s)	Median (s)
1	1.30	1.01
2	1.42	1.02
3	1.43	1.01

(b) Twenty-five fragment publication.

Table 5.5: Descriptive statistics of fragmented publication of a Base64 encoded payload.



(a) Four fragment publication.

(b) Twenty-five fragment publication.

Figure 5.4: Transaction times of Base64 encoded payload.

Results obtained for 4 and 25 fragments publishing with a Base16 encoded payload can be recalled from tables 5.2a and 5.4 respectively. Compared to them, the transmission time for 4 fragments saw a 35.8% decrease and for 25 fragments it saw a 24.6% decrease.

In conclusion, transitioning the encoding of the acquisition data from Base16 to Base64 yielded significant throughput improvements of 30% on average. As such, the conclusion is obvious: the encoding scheme should be altered to Base64.

5.3 MQTT QoS Level

The third question deals with the MQTT message publishing QoS level in use. This section will deal with the QoS level suitability for the equipment operation, its consequences in transmission throughput and eventual side-effects.

As noted previously, the ESP32-S2 client is currently set to publish at QoS level 1. The QoS level can be set as a parameter of the *esp_mqtt_client_publish()* function, as provided by the MQTT client library that is part of the ESP-IDF SDK. As mentioned before, the MQTT standard defines three QoS levels - zero, one and two - each providing increasing delivery guarantees, at the expense of an increased overhead.

QoS 0 is the minimal level as it does not guarantee delivery - the message is either delivered once or not at all. It provides the same guarantees as the underlying protocol, usually TCP.

QoS 1 ensures the message is delivered to the receiver at least once. A message published at this level needs to be assigned an unused Packet Identifier and be considered unacknowledged until a PUBACK packet with a matching Packet Identifier is received. If the sender does not receive a matching PUBACK packet after a given amount of time it should send the PUBLISH packet again. Only after a suitable PUBACK packet is received can the sender discard the message.

QoS 2 is the highest quality of service. This level ensures that no message is lost nor a duplicate is sent. Like at level 1, a message published at QoS 2 needs to be assigned an unused Packet Identifier and must be considered unacknowledged until a PUBREC packet with a matching Packet Identifier is received. When the sender receives the PUBREC, it must send a PUBREL packet

containing the same Packet Identifier in the PUBLISH packet sent. This PUBREL packet must be considered unacknowledged until the sender receives the corresponding PUBCOMP from the receiver. Only afterward can the stored state be discarded [15].

Before taking into consideration any further test results, it was decided that QoS level 0 and its lack of delivery guarantees would not be suitable for the established usage scenarios of the equipment as a whole. As such, further analysis will only contemplate QoS levels 1 and 2.

5.3.1 Experimental Setup

A test program was developed to better assess the impact of the QoS level on the communication efficiency. The program sets up 100 transactions consisting of publishing a number of data fragments and then, depending on the QoS level chosen, either awaits to receive a PUBACK packet for QoS level 1 or waiting to receive a PUBCOMP packet for QoS level 2. As before, to compute each transaction time it time-stamps the first publish command sent and the last confirmation packet received using the *esp_timer_get_time()* function.

Each test batch is comprised of 100 transactions, each containing an acquisition message split in four fragments of 16616 bytes each and also in the transmission of the same amount of data in a single fragment. Each run was repeated twice for both publishing QoS levels 1 and 2.

Each batch of tests mentioned before was run three times and a representative run of each was chosen and included in figure 5.5. Each point corresponds to a transaction and the vertical axis denotes the time in seconds it took to complete, from the first message published to the last acknowledgement packet received.

Table 5.6 collects the mean and median transaction times as an average of three runs for each QoS level and message delivery type combination.

QoS Level	Mean (s)	Median (s)
1	0.26	0.25
2	0.41	0.49

(a) Fragmented publication.

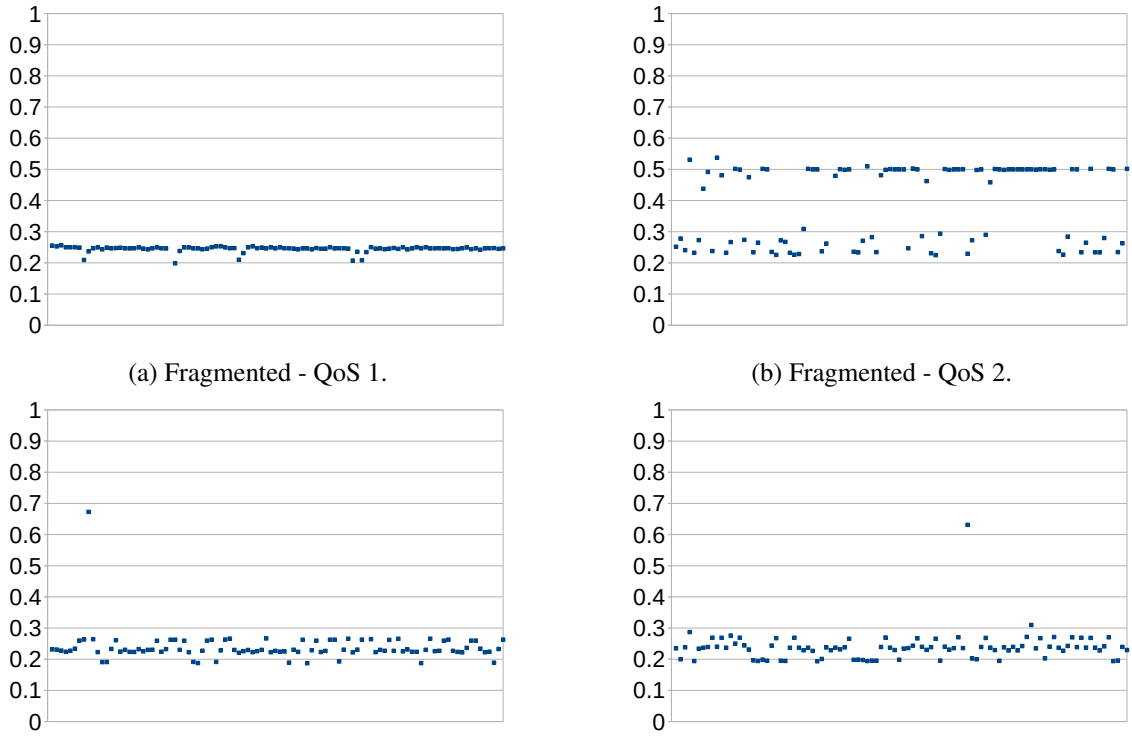
QoS Level	Mean (s)	Median (s)
1	0.24	0.22
2	0.23	0.23

(b) Non-fragmented publication.

Table 5.6: Fragmented and non-fragmented publications at QoS levels 1 and 2.

There is no considerable difference between different message fragmentation options and QoS levels 1 and 2, with the exception of fragmented message publishing at QoS level 2, where both mean and median transaction times saw an increase of 58% and 96%, respectively. Additionally, this setting showed an unstable operation, leading the ESP32-S2 to out-of-memory situations that could only be addressed by enabling the usage of the external PSRAM.

In conclusion, to answer the third question, QoS level 0 is unsuitable for the usage scenarios, because it does not guarantee message delivery. Our study shows that QoS level 2 causes throughput reduction in fragmented publication and QoS 1 leads to a more stable operation both in fragmented and non-fragmented publication. As eventually duplicated transmissions generated



(a) Fragmented - QoS 1.

(b) Fragmented - QoS 2.

(c) Non-fragmented - QoS 1.

(d) Non-fragmented - QoS 2.

Figure 5.5: Transaction time (s) of each run of 100 transactions, fragmented and non-fragmented, for both QoS levels 1 and 2.

at QoS 1 can be dealt with further downstream in the IoT software chain, it is recommended that the messages keep being published at a QoS level of 1.

5.4 IEEE 802.11 Standard

The last question addresses the possibility of a sub-optimal IEEE 802.11 amendment choice, used as a basis for Wi-Fi communication.

Since its introduction in 1997, the IEEE 802.11 standard specifies the protocols used for the implementation of Wireless Local Area Network (WLAN) communication [24]. Many improvements and changes were made through updates to the standard known as amendments. As the ESP32-S2 Wi-Fi subsystem is fully IEEE 802.11b/g/n compliant [19], the throughput analysis will focus on these amendments. Besides other changes, each amendment allows a different maximum theoretical raw data rate. The 802.11b sets the limit at 11 Mbps, 802.11g at 54 Mbps and 802.11n at 150 Mbps [24]. Despite the increasing upper limits, our analysis will show that the technical improvements do not necessarily translate to practical improvements.

In the current setup, the ESP32-S2 device is configured as a wireless client station that connects to a wireless access point. The ESP32-S2 protocol choice can be set using the ESP-IDF function *esp_wifi_set_protocol()*, which currently supports either 802.11b, 802.11b/g or 802.11b/g/n

modes. The device was set to the default mode of 802.11b/g/n. The wireless access point used was also configured through its web interface to the 802.11b/g/n mode.

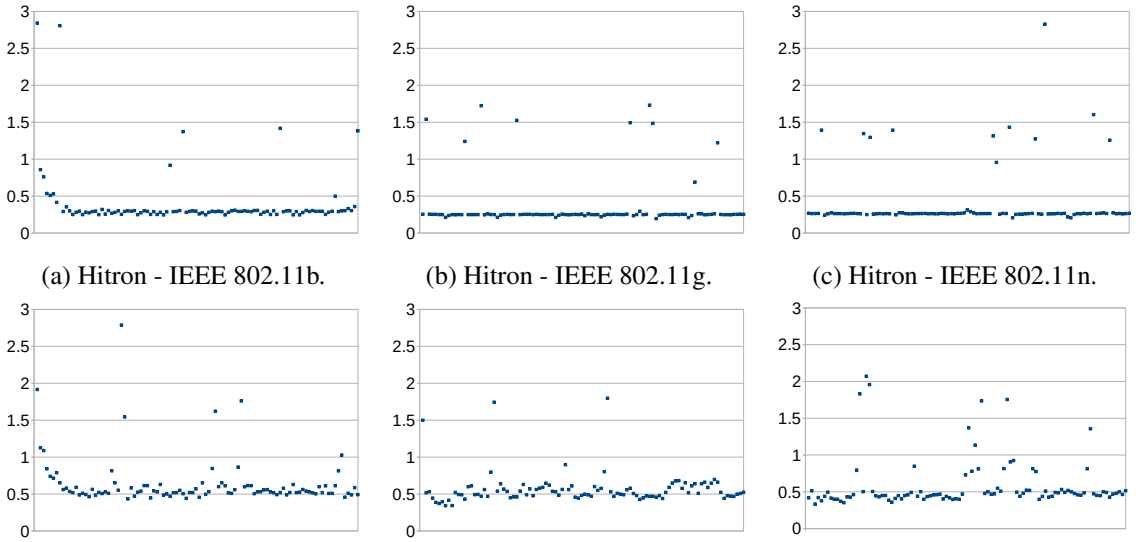
5.4.1 Experimental Setup

A new test program was developed. Like in the previous test program, the messages are published using QoS level 1, meaning it is guaranteed the message is delivered at least once to the receiver. On startup it sets an initialized Wi-Fi interface to one of the supported modes (802.11b, 802.11b/g or 802.11b/g/n) using the *esp_wifi_set_protocol()* function. The program then sets up a number of transactions consisting of publishing a number of data fragments and waiting for a PUBACK packet for each message published. It then computes each transaction time by time-stamping the first publish command sent and the last PUBACK packet is received using the previously mentioned *esp_timer_get_time()* function.

Each test batch consisted of 100 transactions, each containing an acquisition message split in four fragments of 16616 bytes. For each run, a combination of mode definition settings, both on the ESP32-S2 using the *esp_wifi_set_protocol()* function and on the access point through its configuration page was chosen to guarantee the usage of a specific 802.11 amendment.

To ensure the results are representative of different usage scenarios, every test batch was repeated while connected to either a Hitron BVW3653 cable modem router and a TP-Link M7200 4G LTE mobile hot-spot wireless access point. In both cases, the publisher and the broker were connected to the same local network.

Each batch of tests previously described was run three times for each wireless access point and a representative run of each was included in figure 5.6 where each point corresponds to a transaction, and the vertical axis denotes the time, in seconds, it took to complete, from the first message published to the last acknowledgement packet received.



(a) Hitron - IEEE 802.11b. (b) Hitron - IEEE 802.11g. (c) Hitron - IEEE 802.11n.

(d) TP-Link - IEEE 802.11b. (e) TP-Link - IEEE 802.11g. (f) TP-Link - IEEE 802.11n.

Figure 5.6: Transaction time of 100 transactions of four fragments of 16616 bytes.

Table 5.7 compiles the mean and median transaction times as an average of each of the three runs of each access point and 802.11 amendment combination.

Amendment	Mean (s)	Median (s)
802.11b	0.40	0.30
802.11g	0.35	0.25
802.11n	0.58	0.26

(a) Hitron BVW3653 wireless access point.

Amendment	Mean (s)	Median (s)
802.11b	0.69	0.53
802.11g	0.58	0.50
802.11n	0.58	0.47

(b) TP-Link M7200 wireless access point.

Table 5.7: Descriptive statistics of fragmented publications using different IEEE 802.11 amendments.

A simple analysis of table 5.7 shows an expected throughput improvement going from IEEE 802.11b to IEEE 802.11g when the ESP32-S2 is connected to either access point. However, when transitioning to IEEE 802.11n, performance improvements are not so evident. In the case of the Hitron access point, the median transaction time shows even a slight deterioration while the mean time shows a significant increase, indicating the existence of a considerable amount of outlier points in the upper range. The measurements taken with the TP-Link access point show a slight improvement of the median transaction time but no discernible improvement of the mean transaction time, once again indicating the existence of outliers.

To answer the final question, currently, the ESP32-S2 device tries to negotiate the IEEE 802.11n amendment, otherwise, it settles for the IEEE 802.11g or even the IEEE 802.11b if that is not possible. Considering the configuration of the access point currently in usage, the ESP32-S2 does indeed follow this mode setting order starting from IEEE 802.11n. This behaviour was confirmed by inspecting the communication packets using the Wireshark packet analyser.

Moreover, the prioritization of the IEEE 802.11n standard over the IEEE 802.11g showed either no discernible benefit or a vast increase in the mean transaction time, resulting from a growth in the magnitude of the upper range outliers. As such, it is recommended that the ESP32-S2 should restrict the protocol mode to IEEE 802.11b/g, and the wireless access point should be configured accordingly.

5.5 Conclusions

In this section, we summarize the recommendations for optimization which were formulated from the conducted experiments.

From the top of the stack, we were able to conclude that as it is currently done, the fragmentation of data does not incur a significant loss of throughput and is a good trade-off between time efficiency and memory footprint. As a result, it is recommended that the payload remains fragmented as is.

The current encoding scheme of the acquired data was found to be sub-optimal due to its data efficiency of 50%. It is recommended that the encoding scheme should be changed to Base64 as it provided, on average, a 30% throughput improvement.

QoS level 1 used for message publishing was found to be adequate, as level 0 would be unsuitable due to lack of delivery guarantees and level 2 caused decreased throughput and stability. As a consequence, it is recommended that the QoS level for message publishing should continue to be set to 1.

The usage of the IEEE 802.11n standard showed either no benefits or even increased transaction times compared to the IEEE 802.11g protocol. It is recommended that the ESP32-S2 should restrict the protocol mode to IEEE 802.11b/g and that the wireless access point should be configured accordingly.

Finally, it is now possible to confirm that, given the high transmission times of the external communications, the impact of the internal (SPI) communications is negligible. Nevertheless, the recommendations for optimization of the SPI channel still hold.

Part II

Development of a Laboratory Microplate Shaker

Chapter 6

Laboratory Orbital Shaker

This chapter focuses on the development of a laboratory orbital shaker device, namely on component selection, microcontroller firmware and the developed user interface.

6.1 Context

The need for this device arises from the need to process an increasing number of biological samples, and from the fact that when waiting to be processed, the samples need to go through a homogenization step in order to provide accurate reading results. Given the high price of commercial shakers and the lack of models with adequate M2M interfaces, more appropriate for lab automation, it was decided to develop a simple model addressing the project needs.

6.2 Market Research

To better understand the current offerings and capabilities of commercial laboratory shakers, some were chosen for analysis, specifically those that perform an orbital motion in order to match the high-level system requirements. Additionally, features including the capability to holding microplates, having external communication interfaces and a relatively high rotation speed (>500 RPM) were favored for the choice of equipment.

6.2.1 Overview

Taking those features into consideration and after a comprehensive market research, three commercial laboratory shakers were chosen for a deeper analysis.

The first one is the Thermo Scientific Digital Plate Shaker orbital shaker [3], shown in figure 6.1. The Digital Plate Shaker is specifically geared towards shaking microplates and is able to hold six of them in its base plate. As a consequence, its orbit radius is small and its rotation speed range is considerably more extensive while having a higher upper limit compared to general-purpose laboratory shakers. It is only able to be operated locally through a numeric display and tactile switches, offering no means for external automation.



Figure 6.1: Thermo Scientific Digital Plate Shaker [3].

The second one is the IKA MTS 2/4 digital microplate shaker [4], shown in figure 6.2. The IKA MTS 2/4 can hold either 2 or 4 microplates simultaneously. Being a smaller and lighter alternative compared to the aforementioned Digital Plate Shaker, it features a smaller base plate and reduced load shaking capabilities. Moreover, it is also only able to be operated locally through a numeric display and digital switches for setting the timer functionality and a control knob to set the rotation speed, having no means for process automation.



Figure 6.2: IKA MTS 2/4 digital microplate shaker [4].

The third and final one is the BioShake 3000 elm from QInstruments [5], shown in figure 6.3. The BioShake 3000 elm is also specifically geared towards shaking microplates and is able to hold a single microplate at a given time. While having a smaller base plate and a lower load weight rating, its rotation speed is the highest of the three, reaching upwards of 3000 rpm. It is also the lightest of the three and the only one needing an external power supply. It has the specificity of only being able to be operated remotely via a serial interface, making it ideal for laboratory automation purposes. As a consequence, conventional laboratory usage becomes cumbersome and impractical due to having no simple way of being controlled and no local way to display its operational status.



Figure 6.3: QInstruments BioShake 3000 elm [5].

The previous overview is simultaneously summarized and expanded with additional specifications and concrete numerical values in table 6.1.

These three examples will be used to define the requirements for the shaker to be developed. The prototype should be 3D printed and a suitable motor will be chosen, together with an appropriate control system and user interface.

Table 6.1: Comparison overview of selected shaker devices characteristics.

Make/Model	Thermo Scientific Digital Microplate Shaker	IKA MTS 2/4 digital microplate shaker	QInstruments BioShake 3000
Platform holders	Microplate	Microplate	Microplate
Max. no. of microplates	6	4	1
Max. platform load	1 kg @1200 rpm 4 kg @ <400 rpm	1 kg	500 g
Mixing motion	Orbital	Orbital	Orbital
Orbit	2.5 mm	3 mm	2.0 mm
Rotational speed range	150 - 1200 rpm	50 - 1100 rpm	200 - 3000 rpm
Device control	Digital	Digital	Digital
Operation control	Local only	Local only	Remote only
Local interface	Numeric display, switches	Numeric display, switches, control knob	-
Remote interface	-	-	TIA 232
Input voltage	100-240 VAC, 50/60 Hz	220-240 VAC, 50/60 Hz	24 VDC
Power supply	Internal	Internal	External
Net weight	12.8 kg	3.2 kg	1.5 kg
Dimensions	383 x 260 x 145 mm	185 x 105 x 320 mm	142 x 99 x 48.2 mm
Ambient operating temperature	5 ºC to 40 ºC	5 ºC to 40 ºC	5 ºC to 45 ºC
Ambient operating relative humidity	20% to 90%	Up to 80%	Up to 80%

6.2.2 Device Requirements

As a result of the conducted market research and technical discussions with the company, a few key system requirements were set to guide the development of the new orbital shaker. As such, the device should:

- Hold one 96-well microplate.
- Have a 5 mm orbital radius.
- Have a rotational speed range of 200 to 2000 RPM.
- Be digitally controlled.
- Support a remote and a local interface.
- Feature an external power supply.

This set of requirements will guide the next steps of the development of the orbital shaker, ranging from the choice of components, the developed interfaces and control approaches.

6.3 Choice of Components

6.3.1 Microcontroller

A microcontroller needed to be selected to host the control interfaces, the program logic and to set up and interact with the motor control board. Considering the time constraints and previous knowledge, the microcontroller platforms considered viable for selection were the Arduino AVR boards, ARM-based boards or an ESP32-S2 board from Espressif.

The differentiating factor between the previous three options was the well-integrated Wi-Fi stack of the Espressif platform, as well as the versatile accompanying libraries and frameworks. These remote communication capabilities would reveal to be particularly useful to implement a remote user interface. Specifically, the ESP RainMaker framework was [25] revealed to be a straightforward way to create remote graphical user interfaces for Android and iOS. Besides that, the Wi-Fi stack and related protocol libraries leave possibilities open for future features and use-cases.

The previous knowledge of the base libraries and some protocol libraries that are part of the ESP-IDF SDK and having already set up the development environment allowed for the development to start smoothly. Another important aspect is the SDK close integration with FreeRTOS, which gave access to many built-in protocol libraries and the usual benefits of a Real-Time Operating System (RTOS), such as managing concurrent tasks, defining exclusive memory access, defining software timers, among others.

As a result of the above considerations, the small-sized ESP32-S2-Saola-1 development board [6], shown in figure 6.4, was chosen. On the software side, the ESP-IDF SDK, FreeRTOS and the ESP RainMaker framework were used.

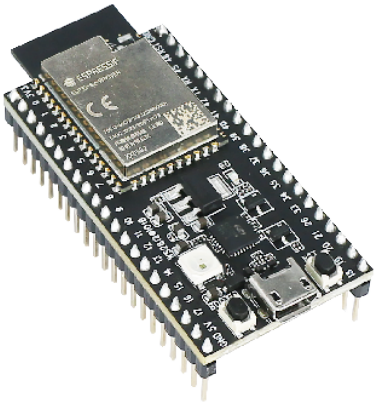


Figure 6.4: Espressif ESP32-S2-Saola-1 development board [6].

The complete device concept can be sub-divided into a few key components and interfaces that encompass its overall functionality surrounding the microcontroller unit, as shown in figure 6.5.

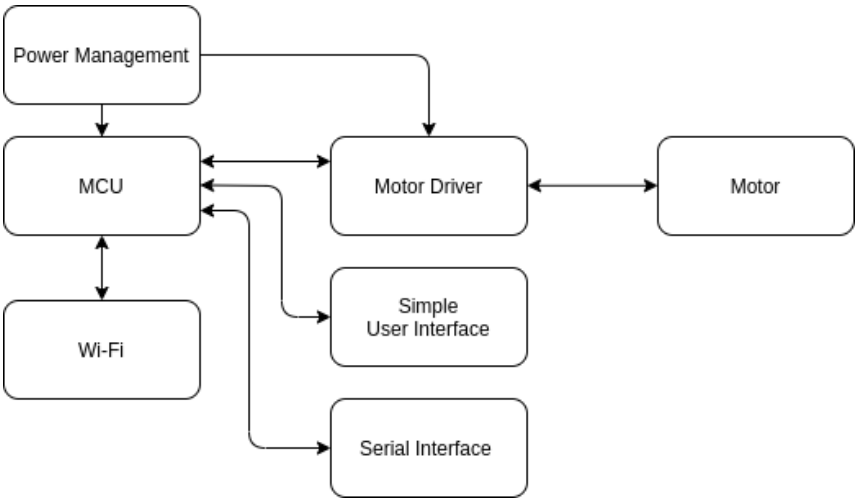


Figure 6.5: Shaker device system concept.

6.3.2 Motor

The rotational movement of the motor is converted into orbital movement by an adequate, 3D printed mechanical structure.

After a few practical experiments with an already available stepper motor, it was clear that despite its perceivably high maximum torque, its rough movement and low maximum rotation speed would not fit the orbital shaker application.

The smooth and continuous rotation at high-speed requirements made brushless DC motors the best solution for this application.

Due to time constraints, motors readily available on European resellers were preferred for a lower lead time. This requirement decreased the pool of available motors reasonably suited for

the application to the BLDC line of motors by ACT Motor. All of them featured a 3-phase, 8 pole design with a rated voltage of 24V and integrated Hall sensors. Still, they all featured different peak torque, rated current values and physical dimensions.

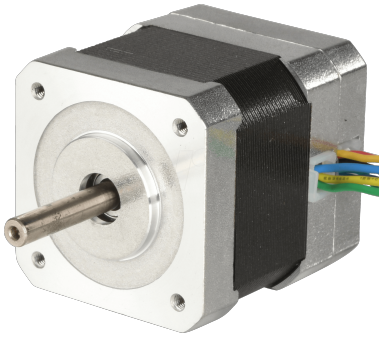


Figure 6.6: 42BLF01 brushless motor [7].

Considering the expected current ranges of the controller boards to be selected as well as the motor dimensions, the ACT Motor 42BLF01 brushless motor shown in figure 6.6 was chosen.

6.3.3 Motor Controller Board

For reasons of efficiency of work, it was decided to use an integrated control solution for the control of the brushless DC motor, instead of developing everything from scratch. The high-level requirements consisted of a considerable range of input speed control methods, such as PWM, I2C or analog voltage, input voltages compatible with those of the selected microcontroller, a small physical footprint and an order lead time compatible with the time allocated for this work.

Having a reasonable order lead time considerably restricted the pool of available controller boards, limiting the choice to European suppliers. The requirement of a small footprint also excluded most high-power controller boards, usually geared towards heavier industrial applications.

These limitations narrowed down the possible candidates to the Brushless Click line boards by Mikroe [26] as those control boards featured a small footprint, were suitable for lower-power applications and could be sourced either through European resellers or through Mikroe itself. Unlike other alternatives, clear specifications, circuit schematics, 2D and 3D models and software examples where applicable are supplied for each board.

The search was then reduced to two similarly sized control boards: the Brushless 7 Click and the Brushless 8 Click, both by Mikroe. Both these boards offered suitable speed control input methods such as PWM, I2C and analog voltage, but the Brushless 8 Click input voltage levels did not match those of the microcontroller and the motor input voltage range (6.5-22V) limited the power supply and motor choice compared to the Brushless 7 Click (11-27V).

For these reasons, the choice was in favor of the Mikroe Brushless 7 Click control board [8], shown in figure 6.7. The main IC, the predriver for 3-phase brushless motor TC78B009FTG, from Toshiba [9], is located on the top side of the board and some of its pins are broken out for ease of access. Also on the top side, four micro-switches allow to easily set some common configuration

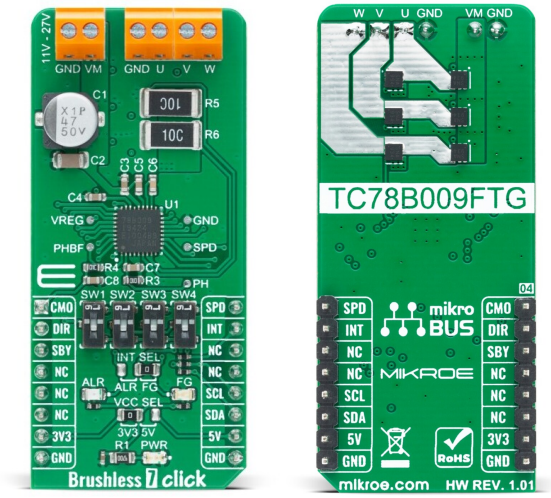


Figure 6.7: Mikroe Brushless 7 Click control board [8].

options. On the bottom side, 6 SSM6K513NU N-Chanel MOSFETs [27] drive the motor phases, up to a maximum motor current of 5A. Due to their placement, a heat sink can be easily added, if needed.

6.4 Interfaces

As the orbital shaker should be able to be integrated into a fully automated laboratory workflow, remote interfaces were preferred. For that, both a serial interface and a smartphone interface were developed. These interfaces can be simultaneously independently used of each other, as a congruent internal state is maintained between the two of them.

These interfaces and their integration with the control logic are managed with the aid of a Real-Time Operating System. In particular, the Espressif fork of FreeRTOS. This operating system integrates well with the ESP-IDF development framework and libraries. The RTOS deals with task management support, shared resources access management and provides software timers to trigger low-frequency events.

A Hardware Abstraction Layer (HAL) comprising the hardware functionality was developed to compartmentalize the hardware and peripheral access from both the internal and external input sources.

6.4.1 Machine-to-Machine Interface

The machine-to-machine interface is implemented by a task responsible for dealing with serial communication. One of the two ESP32-S2 Universal Asynchronous Receiver/Transmitter (UART) hardware peripherals is initialized and configured for communication at 115200 bps, 8 data bits, no parity and one stop bit. This interface is accessed through a wired, USB serial connection. An interrupt-based approach was taken for event-handling, which is reported using the FreeRTOS

queue functionality. These events are split into data events, which are triggered during normal data reception and several error-related events, such as frame errors and buffer-full reporting.

Certain data events are mapped to either set or get the status of certain actions such as starting and stopping the motor, setting and getting the rotation direction, setting and getting the desired rotation speed and setting a timer for timed operation. All the received data events always return a response, whether it is a generic acknowledgment, the requested data or an error response. Table 6.2 summarizes the usage and responses of the implemented commands.

Table 6.2: Available serial commands and responses.

Command	Response	Error Response	Notes
startShake<CR><LF>	K<CR><LF>	E<CR><LF>	Enables the motor.
stopShake<CR><LF>	K<CR><LF>	E<CR><LF>	Disables the motor.
shakeState<CR><LF>	{0, 1}<CR><LF>	E<CR><LF>	Returns the motor state. 0 = Off, 1 = On
rotateCW<CR><LF>	K<CR><LF>	E<CR><LF>	Sets the rotation direction to clockwise.
rotateCCW<CR><LF>	K<CR><LF>	E<CR><LF>	Sets the rotation direction to counterclockwise.
directionState<CR><LF>	{0, 1}<CR><LF>	E<CR><LF>	Returns the rotation direction. 0 = CW, 1 = CCW
setTargetSpeed<no.><CR><LF>	K<CR><LF>	E<CR><LF>	Sets the target rotation speed.
getTargetSpeed<CR><LF>	<no.><CR><LF>	E<CR><LF>	Gets the target rotation speed.
shakeTimer<no.><CR><LF>	K<CR><LF>	E<CR><LF>	Sets a timer, in seconds, for the shaking process.

This interface can be used in two ways: programmatically, which allows for a fully automated remote operation driven by a script, or manually through a serial terminal, allowing for direct control by an operator.

The manual approach can be taken through the usage of any serial terminal computer applications, as shown in figure 6.8, as long as the baud rate, data bits, stop bits, parity and line endings (CR+LF) are properly configured. This approach is particularly suitable for testing, one-off operations or operations with rapidly changing requirements.

The fully automatic approach can take advantage of serial communication libraries available for most programming languages. After proper initialization and configuration, the device can be controlled using the same serial commands with great flexibility. This approach is particularly useful for scheduled or repeated operations, long command sequences and unattended operation.

While not meant to be a reference for proper control implementation, listing B.1 in appendix B is a simple Python script, leveraging the *pyserial* library, that exemplifies bidirectional serial communication with the device.

6.4.2 Human-Machine Interface

In addition to the machine-to-machine interface, an human-machine interface was developed to conveniently control the orbital shaker via Wi-Fi, using either an Android or an iOS device. This

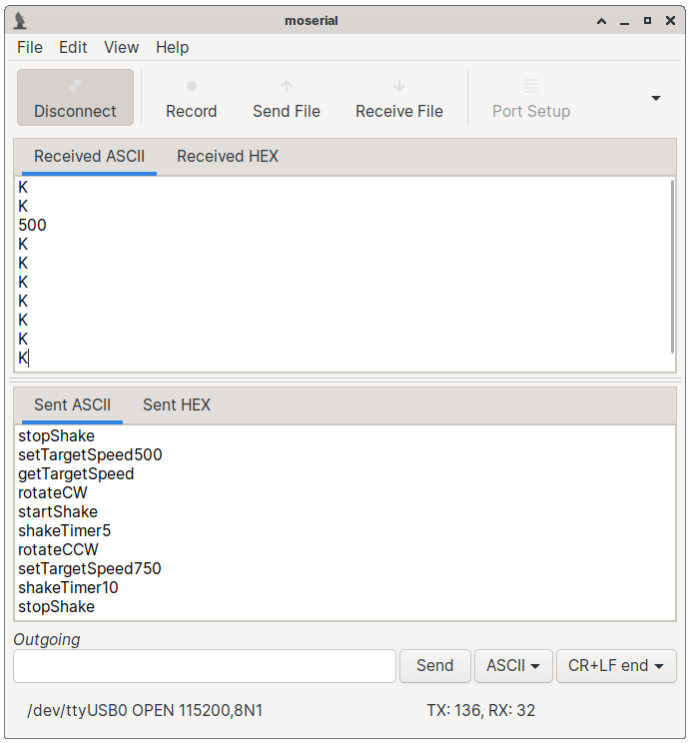


Figure 6.8: moserial, a serial terminal program.

interface was built upon the ESP RainMaker framework from Espressif, available for the ESP32-S2 microcontroller used. Most of the programming and user interface (UI) elements definitions are done in the microcontroller device, while the Espressif RainMaker mobile application is used as a provisioning method, for device management and user interaction.

The ESP RainMaker platform allows for the creation of devices that then aggregate one or more parameters used as a control interface since they are mapped to different UI elements. On the mobile application side, they can be represented as toggle switches, drop-down menus, numerical sliders or text boxes.

The designed interface consists of a toggle switch for the power state, that conceptually maps to the *startShake* and *stopShake* serial command, a drop-down menu that sets the rotation direction, similar to the *rotateCW* and *rotateCCW* commands, two sliders for controlling the rotation speed and timer value as done by the *setTargetSpeed* and *shakeTimer* commands, respectively. A screen capture of the interface is shown in figure 6.9.

Changes to the UI are reported and handled using a callback function, registered on initialization, depending on which command was received. From there, the corresponding HAL functions are called, the internal parameter states are updated and then reported to synchronize the local and remote state. Since this interface can run alongside the serial communication interface, special attention was given to the synchronization of local changes even when a serial interface command was the source of the change in order to keep the mobile interface up-to-date. Additionally, it was ensured that the serial interface remains operational independently of the status of this interface.

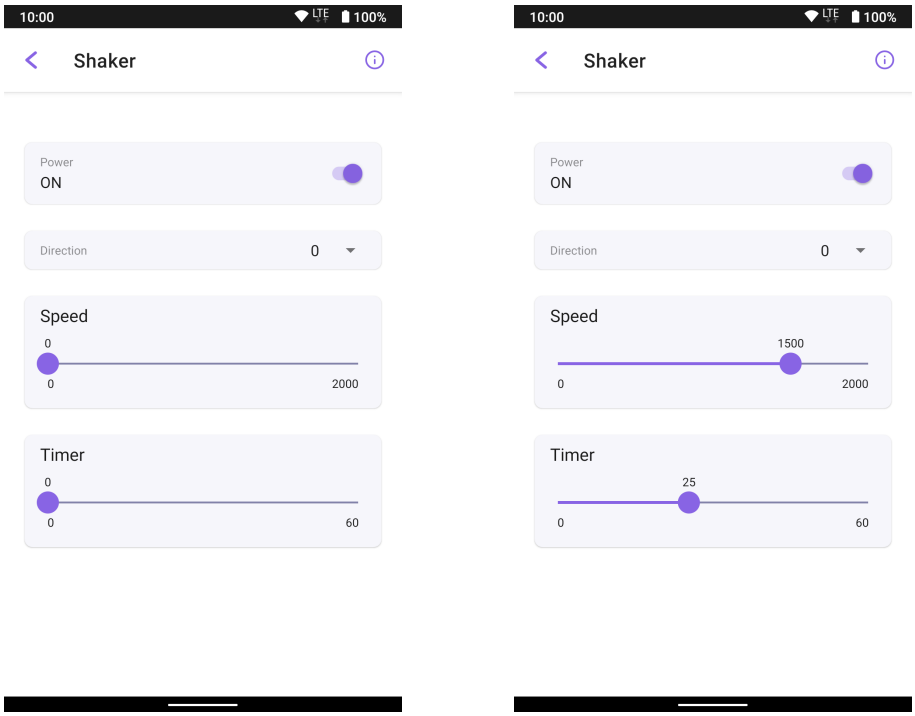


Figure 6.9: The ESP RainMaker-based mobile remote interface.

6.5 Control Approach

The Toshiba TC78B009FTG present in the Mikroe Brushless 7 Click board is a highly configurable three-phase brushless motor sensorless controller. The controller has over 80 configurable settings and while most of them can be written to a given register address using I2C, some can also be set through the micro-switches on the board or directly through the exposed pins.

One of the most relevant signals to set is the SPD signal, which directly controls the rotation speed. Initial tests included setting the speed input using the microcontroller pulse-width modulation (PWM) peripheral with varying duty cycle but were later discarded in favour of using I2C directly since the peripheral was already properly configured in order to read and write to the configuration registers addresses. Using the serial interface, the SPD value is set and get by the *setTargetSpeed* and *getTargetSpeed*, respectively. On the mobile interface, it is defined and retrieved by the *Speed* slider.

The example curve in figure 6.10 shows the various configurable parameters of the speed curve, such as the start duty (1), stop duty (2), end duty (3), start RPM (4), end RPM (5) and slope (6). As a starting point, the speed curve was simplified by setting the starting point to the chart origin and the endpoint to the input duty full range and a speed of 2000 RPM. In order to deal with unfeasibly low RPM values, the SPD lower limit values were clamped before being applied.

The motor rotation direction is set by the combination of both the state of the CWCCW pin and the DIR register setting that reverses the polarity when set. Depending on whether the motor is running when the change direction command is issued, different procedures are taken. If the

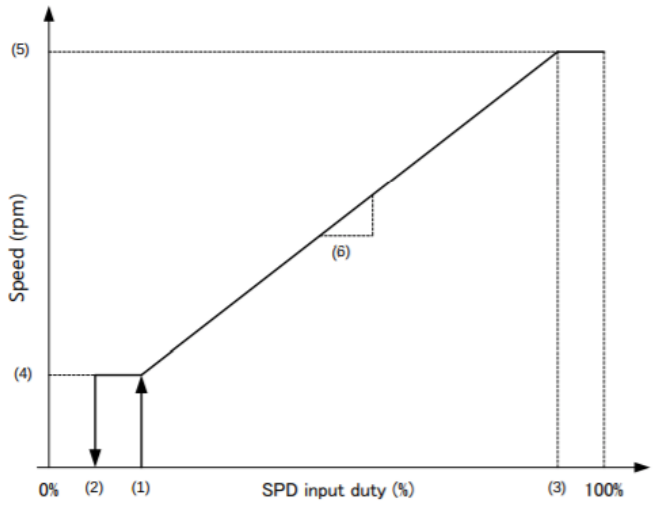


Figure 6.10: Example controller speed curve [9].

motor is running, a soft stop approach is taken in order to bring the motor to a stop. Then, the DIR register is set and the motor soft starts to the previously set speed value. If the motor is stopped, the new rotation direction is simply set in memory and will take effect the next time the motor rotates. Using the serial interface, the rotation direction is set using the *rotateCW* and *rotateCCW* commands and retrieved with *directionState*. On the mobile interface, the state is bi-directionally updated through the *Direction* drop-down menu.

The timer feature allows for the motor to spin for a configurable amount of time, up to 60 seconds. The timer can be set whether or not the motor movement is enabled, but will only start counting when to motor starts spinning. This feature is implemented in the microcontroller using the software timer implementation provided by FreeRTOS. The timer triggers each second in order to update the time remaining in real-time and stops the motor when finished. Using the serial interface, the desired time interval is set using the *shakeTimer* command. On the mobile interface, the time interval is set using the *Timer* slider which is also updated to reflect the current time remaining.

6.6 Mechanical Assembly

Despite the mechanical assembly of the orbital shaker being outside the scope of this work, this section is included as additional contextual information. This device was designed to be used with an optical table, and that is reflected by its protruding attachment feet, as seen in figure 6.11. A slightly different version was designed without the attachment feet, for use in normal laboratory environments.

As of the writing of this document, the device is yet to be printed and assembled. The structure is still under design and the full prototype will be assembled and tested as soon as the structure

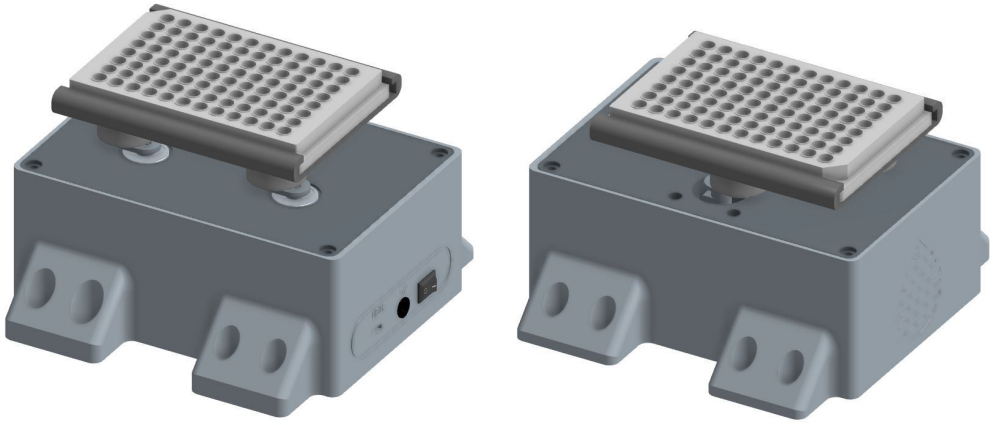


Figure 6.11: Two views of the orbital shaker prototype.

becomes available. The motor and control hardware and software are already fully developed and tested successfully.

6.7 Results

As a result of this work, an orbital microplate shaker, tailored to laboratory automation, was developed. Its shaking capabilities and control logic are on par with the defined requirements. The device uses a small size brushless DC motor, with a sensorless controller and adequate control software. Full control of the motion, in speed, direction and duration, including soft start and soft stop can be achieved either by using the machine to machine interface or the human-machine interface. Table 6.3 summarizes the characteristics of the developed device, alongside the characteristics of the most similar commercial alternative.

Table 6.3: Common and distinguishing characteristics of the developed orbital shaker.

Make/Model:	QInstruments BioShake 3000 elm	Developed Orbital Shaker
Holding system	1 microplate, 500g load	1 microplate, 500g load
Motion	Orbital, 2 mm	Orbital, 5 mm
Rotational speed range	200 - 3000 rpm	200 - 2000 rpm
Device control	Digital, microcontroller-based	Digital, microcontroller-based
Operation control	Remote only	Remote and local
Local interface	-	Smart Phone, via Wi-Fi
Remote interface	Serial TIA232	Serial USB
Power supply	External, 24V	External, 24V
Physical	142 x 99 x 48 mm^3, 1.5 kg	177 x 165 x 74 mm^3, 1.5 kg

It is worth noticing several improved characteristics, not available in the commercial offer, such as a modern serial through USB interface for lab automation purposes and a smartphone-based user interface for quick and easy manual interactions.

Chapter 7

Conclusions

In the first part of this work, we were able to analyze the characteristics of the internal and external communication path of an already developed biological data acquisition device. As a consequence, changes to the device hardware and software architecture were out of reach. To better understand the communication path characteristics, a set of software tests were developed to evaluate the validity of previous assumptions and to study improvement opportunities for various sections of data processing and communication. As a result of the tests, we were able to produce a set of recommendations, alongside relevant software changes, that resulted in the proposed optimization of the device's communication stability and performance.

In the second part of this work, we covered the development of a laboratory orbital shaker, focused on automation and remote control. After conducting a market research and following the input from the company, the system requirements were set to guide the development of the device. Afterward, the components were chosen, and two interfaces were developed. As the main focus was on remote operation, the first interface was a machine-to-machine interface that enables programmatic operation of the device. Additionally, a convenient, smartphone-based human-machine interface was developed to allow easier, direct control of the device for one-off operations.

The following two sections list the contributions resulting from this work and some additional items that can be considered for improvement in the future, respectively.

7.1 Contributions

As a result of the first part of this work, the internal and external communications of the prototype were analyzed and several recommendations were issued towards:

- an improved configuration of the internal SPI communication channel;
- a new and more efficient message encoding scheme;
- keeping the current message fragmentation approach;
- keeping the current MQTT QoS level;

- an improved configuration of the Wi-Fi communication channel.

Additionally, the second part of this work resulted in the design of a remote-operated laboratory orbital shaker, featuring a concurrent machine-to-machine interface and a mobile human-machine interface. To our best knowledge, none of the existing commercial products has this feature.

7.2 Future Work

Regarding the optimizations of the external communications, it would be of interest to further study other aspects that could possibly impact the communication, such as:

- Understanding the effects of using an external antenna (rod or PCB) for the ESP32 board concerning the connection range and throughput - there was no time to address this aspect in the current work.
- Measuring the impact of changing the communication protocol from MQTT to another suitable IoT protocol such as the Advanced Message Queuing Protocol (AMQP), the Constrained Application Protocol (CoAP), or the Extensible Messaging and Presence Protocol (XMPP). This will however imply important changes in other parts of the system, namely at the level of the cloud provider.

As for the orbital shaker device, it would be of interest to implement additional improvements in the interface and in the motor control. Future developments could include:

- an expanded serial command selection, to allow greater controller board configuration and monitoring;
- the development of a preset system, allowing for a number of different settings combinations to be quickly selected for different use-cases;
- the use of Hall effect sensors to enhance the motor control at low speeds.

Appendix A

Base64 Decoding Script

The change of the message encoding scheme in the prototype device implies a different decoding process on the receiving software further downstream. This annex presents a simple implementation of a blocking MQTT client that decodes incoming Base64 encoded messages.

Listing A.1: Example Base64 decoding MQTT client script using the Ruby language.

```
require 'mqtt'
require 'json'
require 'base64'

# Minimal example (blocking)
MQTT::Client.connect('mqtt://192.168.1.5/') do |c|
  # Subscribe to topic /pine
  c.get('/pine') do |topic, message|
    # Print the topic and message
    puts "#{topic}: #{message}"
    # Concert the string to a JSON object
    json = message.to_json
    # Convert the JSON object into an Hash
    h = JSON.parse(message)
    # Print the decoded data key
    p Base64.decode64(h['data'])
  end
end
```

Appendix B

Automation Script

The machine-to-machine interface can be used in an interactive way, driven by a human operator using a serial console, or in a completely automatic way driven by an external program. This annex presents a simple script that illustrates the use of the machine-to-machine interface in a fully automatic way.

Listing B.1: Example automation script using the Python language.

```
import serial
import time

ser = serial.Serial(port='/dev/ttyUSB0', baudrate=115200,
                    bytesize=serial.EIGHTBITS,
                    parity=serial.PARITY_NONE,
                    stopbits=serial.STOPBITS_ONE, timeout=1)
def wr(x):
    ser.write(bytes(x.strip() + '\r\n', 'ascii'))
    data = ser.readline()
    print(data)
    return data
wr("stopShake")
wr("setTargetSpeed500")
wr("getTargetSpeed")
wr("rotateCW")
wr("startShake")
time.sleep(5)
wr("rotateCCW")
time.sleep(10)
wr("setTargetSpeed0")
wr("stopShake")
ser.close()
```

Appendix C

Work Plan

This annex describes the work that has been done, the tasks defined in the work plan as well as their expected timeline. Most of the work was done remotely, some work sessions were held at the premises of the company and the lab work was done at FEUP's premises. Three weekly supervising meetings were held, on average, during most of the work plan.

The envisioned work was broken down into several work packages, each containing actionable tasks. Each task was assigned an estimated duration and some took place in parallel to other tasks of the same or of different work packages. The text reflects the original plan and, where necessary, changes are flagged.

- **WP0**: Preparation
  While not directly related to the dissertation work, this work package encompasses the report writing work done during the PDI course, in the span of 12 weeks. Preparation and structure definition is the first task to be done. Afterward, bibliographic research and report writing will take place for the rest of the duration of the course. The outcome of this WP will be the PDI report. Tasks:

  1. **Preparation and structure** Estimated duration: 2 weeks
  2. **Bibliographic research** Estimated duration: 8 weeks
  3. **Writing** Estimated duration: 6 weeks

- **WP1**: Communication optimizations
  Characterize the current prototype and recommend possible optimization strategies. To better understand the existing code base and the limits of the existing technology, the current SPI-MQTT-based implementation will be studied and improved as much as possible. If deemed necessary, a new, more suitable protocol will be selected and implemented to aid in reducing the transmission time. Tasks:

  1. **Design of a new PCB** Estimated duration: 3 weeks
  2. **Improve existing communication approach** Estimated duration: 5 weeks

   (a) Establish the current baseline
   (b) Understand the current codebase
   (c) Implement improvement opportunities
3. **Implement a more suitable protocol (if needed)** Estimated duration: 4 weeks
   (a) Evaluate the implementation approach
   (b) Implement the new protocol
   (c) Evaluate process improvement

Ex-post note: After discussions with the company, task 1 of this WP was canceled. Given the optimization potential identified in task 2, task 3 was deemed not necessary and was cancelled.

- **WP2**: Microplate Shaker Development
  This work package focuses on the microplate homogenization sub-system. The first task consists of understanding the working principle, then both the hardware (structure, motor control) and microcontroller firmware (control logic, integration) will be developed. Tasks:

  1. **Structure planing** Estimated duration: 2 weeks
  2. **Development** Estimated duration: 7 weeks
     (a) Electrical considerations
     (b) Device communication and user interfaces
     (c) Motor control
  3. **Tests and conclusions** Estimated duration: 3 weeks

- **WP3**: Project Management and Reporting
  This work package focuses on project management tasks and reporting. Time is scheduled for work plan control and for tasks such as report writing and dissertation writing.

  1. **Definition and control of the work plan** Estimated duration: continuous
  2. **Intermediate reporting** Estimated duration: continuous
  3. **Dissertation writing** Estimated duration: 4 weeks

  Contingency measures will be put in place whenever needed to address any unexpected problems as project planning is considered a fundamental aspect for the project's success.

Document version: V0.5, 17-2-2021 — 1st Sem – Preparatory phase (PDI Rules) — Exams 1st Semester

Automatização de plataforma de processamento de amostras biológicas	**iLOF**		(a)				Christmas	New Year			(b)		(c)	Sun
		16 Nov	23 Nov	30 Nov	07 Dec	14 Dec	21 Dec	28 Dec	04 Jan	11 Jan	18 Jan	25 Jan	01 Feb	07 Feb
		W1	**W2**	**W3**	**W4**	**W5**	**W6**	**W7**	**W8**	**W9**	**W10**	**W11**	**W12**	**W12**
WP0: PDI Report				D0.V1										D0.V2
T1: Preparation and structure														
T2: Bibliographic research														
T3: Writing														
WP1: Cloud communications improvements														
T1: Assembly PCB														
T1.1: Familiarize with the process/equipment														
T1.2: Solder the required components														
T2: Optimize SPI communication														
T2.1: Establish the current baseline														
T2.2: Develop a test program														
T2.3: Assess signal quality														
T3: Optimize MQTT communication														
T3.1: Establish the current baseline														
T3.2: Simulation and testing of possible improvements														
T3.3: Understand the current codebase														
T3.4: Implement improvement opportunities														
T3.5: Evaluate process improvement														
T4: Implement HTTP (or other best suited) communication														
T4.1: Evaluate the need for a new protocol														
T4.2: Implement the new protocol														
T4.3: Evaluate process improvement														
WP2: Shaker development														
T1: Requirement analysis														
T2: Development														
T2.1: Structure														
T2.2: Electrical														
T2.3: Communication & User Interface														
T2.4: Motor control														
T3: Tests and conclusions														
WP3: Project Management and Reporting														
T1: Definition and control of the workplan														
T2: Intermediate reporting														
T3: Dissertation writing														

Figure C.1: Gantt diagram for the first phase of the work plan (PDI).

Document version: V0.5, 17-2-2021 — 1st week of second semester

Automatização de plataforma de processamento de amostras biológicas	iLOF	Mon	Carnival	Mon	Mon	Mon	Mon	Mon	Easter					FEUP	
		08 Feb	15 Feb	22 Feb	01 Mar	08 Mar	15 Mar	22 Mar	29 Mar	05 Apr	12 Apr	19 Apr	26 Apr	03 May	10 May
		W1	**W2**	**W3**	**W4**	**W5**	**W6**	**W7**	**W8**	**W9**	**W10**	**W11**	**W12**	**W13**	**W14**
WP0: PDI Report															
T1: Preparation and structure															
T2: Bibliographic research															
T3: Writing															
WP1: Cloud communications improvements															
T1: Assembly PCB															
T1.1: Familiarize with the process/equipment															
T1.2: Solder the required components															
T2: Optimize SPI communication						X	X								
T2.1: Establish the current baseline															
T2.2: Develop a test program						X	X								
T2.3: Assess signal quality															
T3: Optimize MQTT communication								X							
T3.1: Establish the current baseline			?	?	?										
T3.2: Simulation and testing of possible improvements															
T3.3: Understand the current codebase			X	X	X										
T3.4: Implement improvement opportunities															
T3.5: Evaluate process improvement															
T4: Implement HTTP (or other best suited) communication															
T4.1: Evaluate the need for a new protocol															
T4.2: Implement the new protocol															
T4.3: Evaluate process improvement															
WP2: Shaker development															
T1: Requirement analysis															
T2: Development															
T2.1: Structure											**2.1**				
T2.2: Electrical															
T2.3: Communication & User Interface															
T2.4: Motor control															
T3: Tests and conclusions															
WP3: Project Management and Reporting															
T1: Definition and control of the workplan															
T2: Intermediate reporting															
T3: Dissertation writing															

Document version: V0.5, 17-2-2021 — 1st week of second semester — Exams 2nd Semester

Automatização de plataforma de processamento de amostras biológicas	iLOF	Mon													
		08 Feb	17 May	24 May	31 May	07 Jun	14 Jun	21 Jun	28 Jun	05 Jul	12 Jul	19 Jul	26 Jul	02 Aug	09 Aug
		W1	**W15**	**W16**	**W17**	**W18**	**W19**	**W20**	**W21**	**W22**	**W23**	**W24**	**W25**	**W26**	**W27**
WP0: PDI Report															
T1: Preparation and structure															
T2: Bibliographic research															
T3: Writing															
WP1: Cloud communications improvements															
T1: Assembly PCB															
T1.1: Familiarize with the process/equipment															
T1.2: Solder the required components															
T2: Optimize SPI communication															
T2.1: Establish the current baseline															
T2.2: Develop a test program															
T2.3: Assess signal quality															
T3: Optimize MQTT communication															
T3.1: Establish the current baseline															
T3.2: Simulation and testing of possible improvements															
T3.3: Understand the current codebase															
T3.4: Implement improvement opportunities															
T3.5: Evaluate process improvement															
T4: Implement HTTP (or other best suited) communication															
T4.1: Evaluate the need for a new protocol															
T4.2: Implement the new protocol															
T4.3: Evaluate process improvement															
WP2: Shaker development															
T1: Requirement analysis															
T2: Development															
T2.1: Structure															
T2.2: Electrical															
T2.3: Communication & User Interface															
T2.4: Motor control															
T3: Tests and conclusions															
WP3: Project Management and Reporting									**Submit**						
T1: Definition and control of the workplan															
T2: Intermediate reporting															
T3: Dissertation writing									Final rev						

Figure C.2: Gantt diagram for the second phase of the work plan.

References

[1] Lilly_M. English: 96 well plastic cell culture plate, January 2011. URL: https://commons.wikimedia.org/wiki/File:96_well_plastic_cell_culture_plate-03.jpg.

[2] Ubahnverleih. ESP32 Espressif ESP-WROOM-32 Dev Board, November 2018. URL: https://commons.wikimedia.org/wiki/File:ESP32_Espressif_ESP-WROOM-32_Dev_Board.jpg.

[3] Digital Microplate Shakers, 2019. URL: https://www.thermofisher.com/order/catalog/product/88882005.

[4] MTS 2/4 digital microtiter shaker. URL: https://www.ika.com/en/Products-Lab-Eq/Shakers-Vortex-mixer,-Lab-shakers-csp-179/MTS-24-digital-microtiter-shaker-cpdt-3208000/.

[5] BioShake 3000 elm | High-end shaker for microplates. URL: https://www.qinstruments.com/automation/bioshake-3000-elm/.

[6] Espressif Systems. ESP32-S2-Saola-1. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32s2/hw-reference/esp32s2/user-guide-saola-1-v1.2.html.

[7] reichelt elektronik GmbH & Co. KG. ACT 42BLF01. URL: https://www.reichelt.com/us/en/dc-motor-buerstenlos-24-v-26-w-6-3-ncm-act-42blf01-p271490.html.

[8] Mikroe. Brushless 7 Click. URL: http://www.mikroe.com/brushless-7-click.

[9] Toshiba. TC78B009FTG - sensorless pwm predriver for 3-phase brushless motor, March 2020. URL: https://toshiba.semicon-storage.com/info/docget.jsp?did=67874&prodName=TC78B009FTG.

[10] N. Naik. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. In *2017 IEEE International Systems Engineering Symposium (ISSE)*, pages 1–7. IEEE, October 2017. doi:10.1109/SysEng.2017.8088251.

[11] Joana S. Paiva, Pedro A. S. Jorge, Rita S. R. Ribeiro, Meritxell Balmaña, Diana Campos, Stefan Mereiter, Chunsheng Jin, Niclas G. Karlsson, Paula Sampaio, Celso A. Reis, and João P. S. Cunha. An intelligent Lab on Fiber Approach for Human Cancer Single-Cell Type Identification. *Scientific Reports*, 10(1):3171, February 2020. doi:10.1038/s41598-020-59661-5.

[12] Eric Jones, Sam Michael, and G. Sitta Sittampalam. Basics of Assay Equipment and Instrumentation for High Throughput Screening. In Sarine Markossian, G. Sitta Sittampalam, Abigail Grossman, Kyle Brimacombe, Michelle Arkin, Douglas Auld, Christopher P. Austin, Jonathan Baell, Jose M. M. Caaveiro, Thomas D. Y. Chung, Nathan P. Coussens, Jayme L. Dahlin, Viswanath Devanaryan, Timothy L. Foley, Marcie Glicksman, Matthew D. Hall, Joseph V. Haas, Samuel R. J. Hoare, James Inglese, Philip W. Iversen, Steven D. Kahl, Stephen C. Kales, Susan Kirshner, Madhu Lal-Nag, Zhuyin Li, James McGee, Owen McManus, Terry Riss, Peter Saradjian, O. Joseph Trask, Jeffrey R. Weidner, Mary Jo Wildey, Menghang Xia, and Xin Xu, editors, *Assay Guidance Manual*. Eli Lilly & Company and the National Center for Advancing Translational Sciences, Bethesda (MD), 2004. URL: http://www.ncbi.nlm.nih.gov/books/NBK92014/.

[13] ESP32 Modules and Boards. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/modules-and-boards.html.

[14] MQTT Version 5.0, March 2019. URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.

[15] ISO/IEC. ISO/IEC 20922:2016. Technical report, ISO, June 2016. URL: https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/06/94/69466.html.

[16] Tim Berners-Lee. Tim Berners-Lee. URL: https://www.w3.org/People/Berners-Lee/.

[17] Paul J. Leach, Tim Berners-Lee, Jeffrey C. Mogul, Larry Masinter, Roy T. Fielding, and James Gettys. Hypertext Transfer Protocol – HTTP/1.1, June 1999. URL: https://tools.ietf.org/html/rfc2616.

[18] E. Rescorla. HTTP Over TLS, May 2000. URL: https://tools.ietf.org/html/rfc2818.

[19] Espressif Systems. ESP32 Datasheet, April 2020. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[20] Espressif Systems. ESP32-WROVER Datasheet, November 2020. URL: https://www.espressif.com/sites/default/files/documentation/esp32-wrover_datasheet_en.pdf.

[21] Espressif Systems. Application Protocols - ESP32. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/protocols/index.html.

[22] Espressif Systems. Wi-Fi Driver - ESP32. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/wifi.html#esp32-wi-fi-throughput.

[23] Espressif Systems. SPI Slave Driver - ESP32-S2. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32s2/api-reference/peripherals/spi_slave.html.

[24] Dong Chen. A Survey of IEEE 802.11 Protocols: Comparison and Prospective. In *2017 5th International Conference on Mechatronics, Materials, Chemistry and Computer Engineering (ICMMCCE 2017)*, pages 569–578. Atlantis Press, September 2017. URL: https://www.atlantis-press.com/proceedings/icmmcce-17/25882486, doi:10.2991/icmmcce-17.2017.106.

[25] Espressif Systems. ESP RAINMAKER. URL: https://rainmaker.espressif.com/index.html.

[26] MikroElektronika. Brushless Click Boards. URL: http://www.mikroe.com/click?type*=brushless.

[27] Toshiba. SSM6K513NU - silicon n-channel MOS, June 2018. URL: https://toshiba.semicon-storage.com/info/docget.jsp?did=53148&prodName=SSM6K513NU.