Remote Boot Manager for
Raspberry Pi Cluster

João Matos

UMinho | 2019

Universidade do Minho
Escola de Engenharia

João Pedro da Costa Matos

Remote Boot Manager for Raspberry Pi Cluster

October 2019

Universidade do Minho
Escola de Engenharia

João Pedro da Costa Matos

Remote Boot Manager for Raspberry Pi Cluster

Dissertação de Mestrado
Engenharia Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação do
Professor Doutor Jorge Cabral

October 2019

Agradecimentos

Em primeiro lugar quero agradecer aos meus pais e à minha irmã, que no que puderam sempre me ajudaram, foram pacientes comigo e sempre confiaram em mim e foram presentes. E também por tudo o que me ensinaram, muito obrigado, sem vocês não seria possível. Obrigado à minha mãe por ser paciente e querida e por ser minha mãe, ao meu pai por me ajudar e acompanhar em tudo o que faço, e à minha irmã por ser a minha parceira desde que me lembro.

Quero agradecer a toda a minha família, por fazer parte de mim e por me ter visto e ajudado a crescer, e que sempre se preocupou e preocupa.

Quero agradecer a todos os meus amigos que me acompanharam neste percurso.

Um agradecimento especial ao Afonso que me acompanha desde que sei escrever, ao Rossi que faz com que as boleias para Braga sejam mais que boleias, ao Diogo que é como só ele é, e ao Vladyslav, que me acompanhou na prova de fogo que foi o 4º ano, obrigado por estarem sempre presentes. Quero também agradecer à Faísca por fazer parte deste percurso e que continua a fazer parte mesmo depois do percurso.

Um agradecimento especial também aos meus amigos do secundário, que apesar do tempo não deixaram de ser o que eram, em especial ao João Filipe, sabes bem mano.

Aos amigos que fiz em Erasmus, um agradecimento especial também, com quem vivi momentos inesquecíveis e irrepetíveis e que jamais esquecerei. Sobretudo aos integrantes e criadores do covil tuga: Arado, Ruben, Daniela, Flávia e Isabel, obrigado. Entre todos os amigos que fiz que estão agora espalhados pelo mundo, quero agradecer em particular ao Arli e à Melisa.

Quero agradecer também a todos os meus companheiros de futebol, que me ensinaram muito e me ajudaram a crescer. Quero agradecer ao professor Jorge Cabral por me ter dado a oportunidade de fazer este projeto, pela orientação, apoio e confiança.

Também a todos os professores que tive, não só na universidade mas em todo o meu percurso estudantil, obrigado.

STATEMENT OF INTEGRITY

Resumo

A crescente aplicabilidade e integração de sistemas embebidos interconectados (*clusters*) em produtos e sistemas maiores tem vindo a contribuir para um aumento da eficiência e utilidade dos últimos, devido à rapidez de processamento e capacidade de fazer várias tarefas ao mesmo tempo, e até ao seu baixo consumo de energia.

Com o uso destes *clusters*, aquisição de data, comunicação e outras tarefas pequenas mas importantes são executadas mais rapidamente e com mais eficácia. Dado isto, tornou-se óbvio que ser capaz de supervisionar, gerir e controlar esses *clusters* é essencial para assegurar o bom funcionamento de todo o sistema.

Depois de fazer uma pesquisa intensiva em *papers* e produtos que visam gerir e comunicar com vários microcontroladores, a conlusão a que se chega é que nenhum cumpre os requisitos propostos nesta Dissertação, que são comunicar, detetar a ocorrência de erros de arranque e instalar qualquer sistema operativo, a qualquer momento, em cada *Raspberry Pi* do *cluster*.

O objetivo desta Dissertação foi desenvolver um Sistema de Monitorização Central para *clusters* de *Raspberry Pi* que tem em conta principalmente estes três requisitos.

Foi estabelecida uma conexão *TCP/IP* permanente com cada *Raspberry Pi* do *cluster*, para troca de dados e comandos. Também foi desenvolvida uma Interface Gráfica do Utilizador, que mostra informação atualizada sobre todas as *Raspberry PI* do *cluster* e permite uma gestão individual ou coletiva fácil. A Interface Gráfica do Utilizador também faz com que seja possível fazer o *upload* e *download* de qualquer Sistema Operativo para um servidor *FTP*, para mais tarde ser instalado em qualquer *Raspberry PI*.

A integração deste Sistema de Monitorização em produtos já existentes pode ter implicações muito positivas e melhorar eficácia e eficiência, uma vez que o trabalho, tempo e custo de manutenção foram reduzidos.

O sistema completo torna-se mais versátil, uma vez que o *cluster* pode mudar a sua função, ao instalar um Sistema Operativo quando solicitado.

Palavras Chave: *Cluster*, *FTP*, *Múltiplos Sistemas Operativos*, *Raspberry PI*, *TCP/IP*.

Abstract

The increasing applicability and integration of interconnected embedded systems (clusters) in bigger products and systems has been contributing for an increase in efficiency and utility of the later, due to the clusters' fast processing and multitasking abilities, and even their low power consumption.

With the use of those clusters, data acquirement, communication and other small yet important tasks are executed faster and more efficiency. Given this, it has become obvious that being able to supervise, manage and control these clusters is essential to ensure the proper functioning of the whole system.

After doing a thorough research on papers and products that aim to manage and communicate with multiple microcontrollers, the conclusion taken is that none fulfil the requirements proposed in this *Master's thesis*, which are to communicate, detect boot errors and burn a desired OS at any time in each of the cluster's Raspberry Pi.

The aim of this *Master's thesis* was to develop a Central Monitoring System for Raspberry Pi clusters which takes into account mainly these three requirements.

A permanent TCP/IP connection with each of the cluster's Raspberry Pi was established, for data and command exchanging. A GUI was also developed, which displays updated information about each of the Raspberry Pi and allows for a easy management of each of them individually or all together. The GUI also makes it possible to upload and download any OS to an FTP server, to later be burned to a Raspberry Pi.

The integration of this Monitoring System in already existing products can have very good implications and improve performance and efficiency, as the work, cost and time of maintenance have been reduced.

The whole system becomes more versatile, as the cluster can change its role, by burning a different OS on demand.

Keywords: Cluster, FTP, Multiple Operating Systems, Raspberry PI, TCP/IP.

Table of Contents

List of Figures

List of Tables

Acronyms List

App	Application.
BOOTP	Bootstrap Protocol.
DHCP	Dynamic Host Configuration Protocol.
FTP	File Transfer Protocol.
FTPS	File Transfer Protocol over SSL.
GUI	Graphical User Interface.
HDD	Hard Disk Drive.
HTTP	Hypertext Transfer Protocol.
IP	Internet Protocol.
MAC	Media access contro.
OS	Operating System.
RBM	Remote Boot Manager.
RPi	Raspberry Pi.
SD	Secure Digital.
SFTP	Simple File Transfer Protocol.
SMTP	Simple Mail Transfer Protocol.
SSH	Secure Shell.

SSL Secure Sockets Layer.

TCP Tansmission Control Protocol.

TFTP Trivial File Transfer Protocol.

TLS Transport Layer Security.

VNC Virtual Network Computing.

WAN Wide Area Network.

WLAN Wireless Local Area Network.

Chapter 1

Introduction

The increasing need of computing power has brought up a correspondent rise in the using of clusters. A cluster consists of a group of interconnected processing units which purpose is to execute a specific task, behaving like a single major computer [1]. A cluster can be used for many purposes, such as advanced control of robotics and prosthetics [2], hosting databases [3] and sensor, motor and data management tasks [4], among others.

Resulting from the fact of most of the times having to perform a single specific task, clusters are generally exclusive and not easy to afford [5] nor to monitor, meaning that they should be as efficient as possible. With this in regard, the conclusion is that the main disadvantage of clusters is cost, maintenance and monitoring and elasticity/versatility [6] [7]. Considering the importance and impact of clusters on our routines and life quality, it becomes obvious that they should be optimized so we could take the most of them, with as few effort as possible.

The objective is to ease the management of the cluster and to enable it the possibility of having multiple purposes, and thereby reducing its disadvantages. This is going to be achieved by remote booting any OS and detecting boot errors in each of the cluster's microcontrollers. The first measure intends to increase remote control and give versatility to the cluster, the later to improve management and monitoring over the cluster.

The thesis will focus on research and combination of possible methods and technologies to decrease cluster's hardships and conclude with the reflection of expected vs obtained results.

1.1 Contextualization

The current Master Thesis was developed to fulfill the existing needs of the projects developed by University of Minho's Embedded System Researcher Lab.

Multiple projects are currently being developed, and clusters are part of most of them, and it is desirable that they make everything easier rather than bringing further complications. That is where the suggestion for developing a Remote Boot Manager came from.

In a cluster, malfunctioning microcontroller bootings have to be presentially repaired, which is not practical nor viable. So, more than helping develop new projects, the boot manager can be used to improve already finished projects and systems. Furthermore, it can improve the microcontrollers' performance, since their operating system can be lighter and designed to perform a more specific task, and switched to a different one when another task is required, thanks to the option of installing different operating systems on demand.

1.2 Motivation

With the development of technologies the human race tends to get more and more connected everyday. Nowadays networks and IOTs are a very important part of the majority of electronic systems and devices, and so is their integrity and versatility. A significant amount of these networks as well as sensors HUBs consist of microcontroller clusters. These clusters sometimes perform non-critical tasks such as data retrieving through the sensors, of simple communications, but sometimes they perform critical roles, where ensuring every cell of the cluster is working is important. Also, in some applications, the same clusters (or even each cell of the cluster) has to perform a variety of different tasks, which require different resources and tools, and then versatility becomes an important asset. Both these requirements are fulfilled by some products already, which is to be expectable since it is so important. But to provide this is a simple, practical and intuitive way is something that is not found so easily. In this dissertation the objective is to design and build a remote boot manager that can provide both versatility and integrity to a microcontroller cluster, operating remotely, therefore providing an easy and efficient method of operation. The goal is being able to always have information about every cell of the cluster, as well as control over it, by having a permanent established connection. Building a root of trust is not part of this Thesis goals, security was not a concern during the planing nor implementation of the project.

1.3 Objectives

In order to successfully achieve the proposed goal some objectives were set, regarding the requirements and aiming for a better project planning:

- Set system requirements, taking constraints into consideration;
- Design the system as a whole, according to the requirements;
- Divide the system into parts and modules and define their outlines;
- Design of the GUI;
- Definition of the program flow;
- Implement TCP/IP protocol for basic communication with one client;
- Save information on every microcontroller of the cluster in a database;
- Add a microcontroller's information to the database when it connects for the first time;
- Implement option to remove a microcontroller from the database;
- Expand TCP/IP communication to multi client;
- Expand TCP/IP communication for specific command parsing;
- Get updated information from every microcontroller and test their connection periodically;
- Implement file upload and download from FTP server;
- Assign the GUI buttons to their respective functions;

1.4 Dissertation Structure

This thesis is comprised of six chapters: ”Introduction”; ”State of the Art”; ”System Specification”; ”Implementation”; ”Tests and Results”; ”Conclusions and Future Work”, listed in order of appearance.

The ”Introduction” chapter locates the project technologically. It states the goals of the project and its presence and importance in today's society and necessities, as well as what led to the idea of developing it.

The ”State of the Art” chapter develops a search on articles and products related to the system to be developed, focusing on the aspects they have in common, on where they are good already and where they can be better, and on the methods and technologies being used, aiming to find the best aspects of each of them. The role of this chapter in the project is to determine the best solutions that can be applied to meet the proposed goals.

The ”System Specification” chapter describes the project’s characteristics, behavior, and tools used to design and implement it. It also gives an overview of its constituent parts and of how the user interacts with it.

The ”Implementation” chapter explains how the system was implemented and the choices made upon implementing it. It also fully describes it and explains how the system works.

The ”Tests and Results” chapter describes how the tests during development were done and how its results were interpreted.

The ”Conclusions and Future Work” chapter consists of a reflection of the achieved results in light of the proposed objectives. It also includes some considerations and ideas for future plans to improve the project developed.

Chapter 2

State of the Art

The research has been conducted regarding the main points of the product and using specific keywords. It consists of academic papers, market products, articles and forums threads. The objective of this research is to stay aware of what has already been developed, its advantages and flaws, what needs to be improved, the current market necessities and what tools can be used considering the constraints and proposed objectives.

2.1 Research

2.1.1 Internet-based remote control using a microcontroller and an embedded Ethernet [8]

This paper describes a setup that ”consists of two primary elements communicating with each other”[8]. The purpose of the setup is to remotely control a DC motor. It provides total control over the motor and ”visual display of the current position of the motor using real-time sensor data”[8], making it very useful and easy to use.

However, even though there is remote control, this is not the type of control intended to be achieved in this Thesis, a deeper and more low level control is necessary. Furthermore, it is a two element (host and client) connection, instead of a Raspberry Pi cluster.

2.1.2 Remote monitoring system development via Raspberry-Pi for small scale standalone PV plant [9]

Once again, the system is comprised of one host and one client only. This system is very similar to the one analyzed before, but has no control at all and focuses mainly on the monitoring and data retrieval part. Comparing to the former one and with the mentioned metrics in sight, the adding of the GUI is an improvement.

2.1.3 Design and implementation of Remote Terminal Unit on Mini Monitoring Weather Station Based on Microcontroller [10]

This article explains ”the use of a single master - multi slave communication method in a Remote Terminal Unit (RTU) of Mini Monitoring Weather Station Based on Microcontroller.” [10]. The transition from a two to multiple terminal connection is a significant improvement. Master and slave can be interpreted as host and client, respectively.

However there are some issues and limitations in the communication that can be improved. ”While in broadcast mode, the Master sends a command (query) to all slave. In this addressing mode the slave does not send a response to the Master” [10], this is a critical flaw, given that it is essential to properly control all the clients at once, with a command and response protocol.

2.1.4 Universal remote boot and administration service [11]

”This paper discusses advanced network booting methods overcoming the restrictions of traditional PXE/DHCP/TFTP setups.” [11]. In terms of OS options, this paper’s suggestion is the ideal solution, it does not offer any constraint in this aspect, virtually any OS can be installed. Also, it uses the base of multiple boot services and communication protocols to develop a framework with the best aspects of each of them. The following excerpt describes the idea perfectly: ”It suggests a new approach combining existing well established boot services into a more flexible framework for remote booting a wide range of operating systems and maintenance tools.” [11]. It does not have any kind of application layer other than the booting management itself, but this part is already very useful and very well designed.

2.1.5 Dataplicity

Dataplicity agent is a Python based script which allows remote connection to a Raspberry Pi via ”dataplicity.com”. It has multiple advantages and features:

- Abstracts the user from all configurations such as TCP, VPNs, static IPs, port forwarding;
- Provides the user an URL to access their device, instead of using the IP, this ID works everywhere, not on local netowrks only;
- Forward TCP ports including VNC and SSH from the device using Dataplicity to the workstation;
- Run diagnostics and make common repairs easily and practically, with Dataplicity Custom Actions;
- Provides instant access to device telemetry, data transfer statistics, performance data, device network configuration, cpu information and memory utilisation while device is running, with Dataplicity Diagnostics.

When Dataplicity Agent is installed on a device device, it will establish and maintain a secure HTTPS connection to the Dataplicity IoT Router.

When connecting to the Remote Shell via the Dataplicity website, or to the redirected web interface via the device's URL, the connection will be routed between the browser and the device via IoT Router. In practice, this means it is possible to access the devices covered by Dataplicity anywhere that they have a viable internet connection. The traffic is routed using encrypted websocket connections, and is robust enough to be used in instances where the internet coverage is not robust. Because the device itself is the originator of the connection, traditional impediments to remote access (such as NAT, firewalls and dynamic IP addressing) are no longer an issue.

This product is very useful and well done, however it does not give control of the operating system nor of the Raspberry Pi [12].

2.1.6 TOSIBOX

TOSIBOX® Lock 500 is a high-end connectivity device bringing possibilities for customers to manage their operations and to build new IoT solutions. It supports multiple WLAN and WAN features, connections types, ports and input/output alternatives. Its main features are:

- Massive VPN throughput for data consuming applications, end-to-end encryption from Key to Lock, Lock to Lock or Lock to (Virtual) Central Lock;
- Up to 50 concurrent VPN connections;
- Industrial design: robust and fanless enclosure, integrated DIN rail bracket and industrial screw-on DC power connector;
- Integrated WiFi as connectivity method or access point for wireless devices on site;
- TosiOnline™ functionality of automatic reconnection of dropped connections.

In terms of robustness, connection safety and traffic management it is a very good solution, but lacks in types of communication and control of the clients. Is is best put to use in a industrial environment rather than in a applications specific environment [13].

2.1.7 Balena

Balena is a complete set of tools for building, deploying, and managing fleets of connected Linux devices. It provides infrastructure for fleet owners so they can focus on developing their applications and growing their fleets with as little friction as possible.

Balena's tools are designed to work well together as a platform, but the user can also pick and choose the components they need for their project, and adapt them to their particular use case.

Devices in the balena ecosystem run balenaOS, a bare-bones, Yocto Linux based host OS, which comes packaged with balenaEngine, a lightweight Docker-compatible container engine. The host OS is responsible for kicking off the device supervisor. Within each service's container the user can specify a base OS, which can come from any existing Docker base image that is compatible with the device architecture. The base OS shares a kernel with the host OS, but otherwise works independently. The balena device supervisor runs in its own container, which allows us to continue running and pulling new code even if the user's application crashes [14].

2.2 Basic Concepts

2.2.1 TCP/IP Protocol

The communication between each Raspberry and the Central Application is done through TCP/IP, this protocol was chosen because it allows for a simple and standardized communication. The TCP/IP protocol actually consists of a set of communication protocols in the internet and computer networks. This protocol provides an end-to-end (no intermediaries) communication specifying how data should be packetized, addressed, transmitted, routed, and received. The Transmission Control Protocol (TCP) provides flow-control, connection establishment, and reliable transmission of data. This functionality is organized into four abstraction layers, which classify all related protocols according to the scope of networking involved. Those four layers are the Link layer, Internet layer, Transport layer and Application layer. All the devices on this project are connected to the same network, therefore, the Link layer will be looked into with more detail.

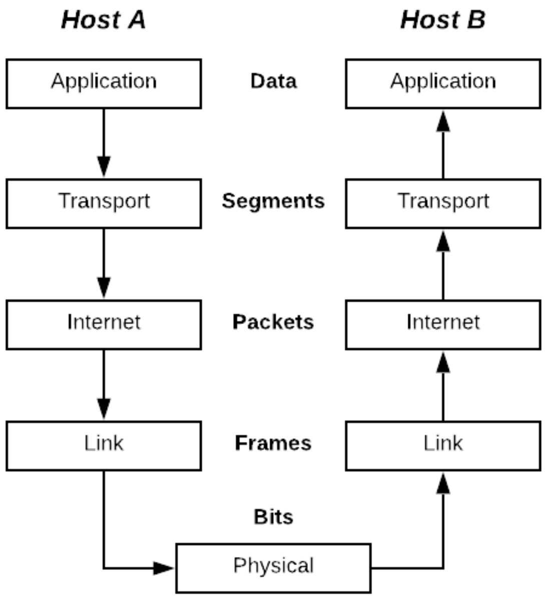


Figure 2.1: Layers and data encapsulation in TCP/IP Protocol

Application Layer

The application layer includes the protocols used by most applications for providing user services or exchanging application data over the network connections established by the lower level protocols.

Examples of application layer protocols include the Hypertext Transfer Protocol (HTTP), the File Transfer Protocol (FTP), the Simple Mail Transfer Protocol (SMTP), and the Dynamic Host Configuration Protocol (DHCP). It is the layer within which applications, or processes, create user data and communicate this data to other applications on another or the same host.

Transport Layer

The transport layer performs host-to-host communications on either the local network or remote networks separated by routers, it provides a channel for the communication needs of applications. The protocols in this layer may provide error control, segmentation, flow control, congestion control, and application addressing (port numbers).

Internet Layer

The internet layer exchanges datagrams across network boundaries. It provides a uniform networking interface that hides the actual topology (layout) of the underlying network connections.The primary protocol in this scope is the Internet Protocol, which defines IP addresses, its function in routing is to transport datagrams to the next IP router that has the connectivity to a network closer to the final data destination.

Link Layer

The link layer is used to move packets between the Internet layer interfaces of two different hosts on the same link. This layer includes the protocols used to describe the local network topology and the interfaces needed to effect transmission of Internet layer datagrams to next-neighbor hosts. There are various processes of transmitting and receiving packets on a given link. These perform data link functions, then actually transmit the frame over a physical medium. TCP/IP model includes specifications of translating the network addressing methods used in the Internet Protocol to link layer addresses, such as Media Access Control (MAC) addresses. This is also the layer where packets may be selected to be sent over a virtual private network or other networking tunnel. In this scenario, the link layer data may be considered application data which traverses another instantiation of the IP stack for transmission or reception over another IP connection [15].

2.2.2 FTP

A FTP server is a server that allows for a collective storage and access of files, through a network, using the File Transfer Protocol (FTP). This server can be public or private (demanding username and password for file access). The server can be accessed through a web browser or a dedicated FTP client [16].

There are some variations of the FTP protocol, with small differences, that may adapt better to some projects than others:

- **FTPS** - There are two kinds of FTPS, implicit and explicit. The implicit type required the use of either a SSL or a TLS connection. The explicit type allows clients to request the FTP session to be encrypted;

- **SSH FTP** - This protocol is very similar to the ordinary FTP, but uses the SSH protocol to transfer files, encrypting both commands and data;

- **TFTP** - TFTP is a simpler version of FTP, it is easier to implement, is lighter and lacks security. It is generally used in early stages of remote booting from a local area network;

- **SFTP** - This variant of FTP was considered unsecured is almost never used [17].

The hosting of the OS images was done through an FTP server. This was the chosen option because it is a simple and efficient way of hosting and accessing files through the internet, in a controlled way. Plus, there is a lot of support on how to use it.

2.2.3 Remote Booting

Remote booting a machine consists in booting it without relying on local resources (hard drive), but instead getting those resources from a server or other specific place in a network [18]. This is very useful to centralize management of disk storage, therefore reducing maintenance costs and time spent. Furthermore, money is saved on Hard Disk Drives (HDD), since there is no need for big storage capacities.

The first stage of remote booting consists on (generally) using the DHCP or BOOTP protocols to establish a connection to the server which a minimal image is coming from. This minimal image enables the partitioning and formatting the hard disk and then downloading, installing and launching the OS [18] [19]. In this project, the referred first minimal image is already permanently stored in the Raspberry Pi SD Card.

2.3 Use Case

2.3.1 SustIMS [20]

In order to better understand the system and its purpose, a useful and practical use case is presented. The case under study is the "Sistema ciber-físico para deteção de colisões"[21], a master thesis developed in Universidade do Minho by Luís Pereira.

This thesis focuses on a refactoring an already existing system called SustIMS, which objective is to detect highway car accidents, to get faster medical assistance and to prevent further chain accidents and traffic jamming. This refactoring consists in redesign the sensor node and the coordinator of all the sensors in order to support Contiki OS, and to implement the 6LoWPAN protocol in the communication between devices.

In short, the product consists of a wireless network of microcontrollers (cluster) that behave has sensor nodes. Taking into account the fact that these nodes are placed in the highway it becomes obvious that maintenance is not easy, as a permit is necessary to stop in the middle of a highway, for example, because it is forbidden to do so without one. Adding to this, the simple fact of having to go to the place where the sensor is consumes considerable time and money and reduces maintenance efficiency. In the case there is an accident, traffic will also be an issue. Considering all of these factors it becomes obvious that a remote handling of software malfunction issues turns out to be very profitable and practical.

This thesis aims to mitigate all of these issues, as it allows for a remote managing of a microcontroller cluster from one central computer, providing a GUI to facilitate this process.

2.3.2 HoneyCloud

"Globally there are more honey bees than other types of bee and pollinating insects, so it is the world's most important pollinator of food crops. It is estimated that one third of the food that we consume each day relies on pollination mainly by bees, but also by other insects, birds and bats." [22]. This perfectly describes the importance of bees to the planet and to the human race, economically, but most importantly, for our survival and subsistence.

Human-made beehives are a very significant factor in the amount of bee colonies. They potentiate the nesting, reproduction and settling of bee colonies, which in their turn potentiate honey production, pollination and also biodiversity and the well faring of ecosystems [23].

Taking these factors into account it becomes clear that the good use and maintenance of beehives is indispensable for our well being and life quality. What happens too often with beekeepers is that they do not know when the time is ripe for collecting the honey from the beehives. If the beekeeper does it too soon, it will kill bees, stress them and also lose honey, in addiction, he will use up resources such as time and money in going there to gather the honey. If he gathers the honey too late the bees start consuming the honey and the damage is mostly monetary [24].

There is a project being developed in the Julius-Maximilians Universität Würzburg that focuses on mitigating that problem. The objective is to "provide an IT infrastructure that allows continuous monitoring of the bees without direct intervention" [31], it uses a set of sensors (weight, humidity,temperature, ...) connected to a microcontroller (sensor HUB) in each beehive to monitor its state. All of these microcontrollers are all connected to a central database which then provides the information to a GUI/phone application.

If there is some issue with one of the microcontrollers someone must go there to detect the problem and then fix it, which consumes time, money, and once more stresses and kills the bees and decreases honey production. Using the Remote Boot Manager to detect these failures and reinstalling the OS on the respective microcontroller could easily solve the problem on most occasions with little effort and no resource wasting.

Chapter 3

System Specification

3.1 System Requirements

3.1.1 Functional Requirements

The functional requirements specify what the system does, its features. [25]

- **Ensure updated information at all times** - The GUI must always have the most recent information on every Raspberry Pi of the cluster, including current installed OS, MAC and IP address, connection state and boot status;
- **Control the cluster remotely** - This includes having remote control of each Raspberry Pi of the cluster, regardless of its OS;
- **Manipulate the OS of every Raspberry Pi** - More than asking to do specific task, have total control of its operating system;
- **Manage FTP server** - Upload and Download any OS image to the FTP server;
- **Remove Raspberry Pi from cluster** - Being able to remove a any microcontroller from the cluster, remotely;
- **Maintain multiple connections** - The Central Application host must be able to maintain established connections to all Raspberry Pi in the cluster.

3.1.2 Non-Functional Requirements

The non-functional requirements are characteristics of the system and how it works. [25]

- **Provide a GUI** - Supply the user with an easy and intuitive way of managing the cluster;
- **Provide multiple OS alternatives** - Ensure there is a way for the Raspberry Pi to be able to burn a different image on itself;
- **Send data exclusively** - Keep the information private by sending the data only to a specific place.
- **Detect Errors** - Raspberry Pi booting errors are detected.

3.2 Constraints

Constraints are conditions that a system must meet and cannot be altered.

- **Support Raspberry Pi** - The system must be compatible with the Raspberry Pi.
- **Connection Limit** - The system must be capable of handling a significant amount of connections.

3.3 Hardware Specification

3.3.1 Raspberry Pi 3 Model B+

The Raspberry Pi is a series of small single-board computers that can be used to perform almost any computing and processing task. It has peripherals for the most commonly connection types, a 64-bit quad core Cortex-A53 processor running at 1.4 GHz and Bluetooth/BLE, among other features, which makes it a small and powerful computer, very used not only for learning purposes[26] but also for performing small, specific tasks.

Specifications

- Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz;
- 1GB LPDDR2 SDRAM;
- 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac
- wireless LAN, Bluetooth 4.2, BLE;
- Gigabit Ethernet over USB 2.0 (maximum throughput 300 Mbps);

- Extended 40-pin GPIO header;
- Full-size HDMI;
- 4 USB 2.0 ports;
- CSI camera port for connecting a Raspberry Pi camera;
- DSI display port for connecting a Raspberry Pi touchscreen display;
- 4-pole stereo output and composite video port;
- Micro SD port for loading your operating system and storing data;
- 5V/2.5A DC power input;
- Power-over-Ethernet (PoE) support (requires separate PoE HAT).[27]

3.4 Software Specification

3.4.1 Qt

Qt is a free and open-source widget toolkit for creating graphical user interfaces as well as cross-platform applications that run on various software and hardware platforms such as Linux, Windows, macOS, Android or embedded systems [28].

QT is most commonly used to develop GUIs and multi-platform applications that run on a variety of desktops and embedded systems. It has many case-specific libraries and is mostly used for software developing in C++ or Python languages [29].

Because of its GUI development focus, developing a GUI with Qt becomes very easy and intuitive, with a lot of libraries and support on the internet. Plus, it is an IDE that had been worked with before. Adding all these factors, the choosing of the IDE for developing the Central Application became easy.

3.5 System Architecture

3.5.1 System Overview

The following image []3.1] serves to better understand the whole system and where this Thesis stands. Its main three elements are the Central Application (this Thesis subject), the FTP server and the Raspberry Pi cluster.

The Central Application (RBM) is in contact with both the cluster and the FTP server. The contact with the FTP server consists of an exchange (Upload an Download) of files (OS images), this is basically setting up the FTP server to be a functional OS database for the cluster to remote boot. The information exchange between the Central Application and the cluster's Raspberry Pi consists of simple messages for requests (information, status) and commands (new image burning, command line commands). The contact between the cluster's Raspberry Pi and the FTP server is unilateral, the microcontrollers download the image to be installed from the server.

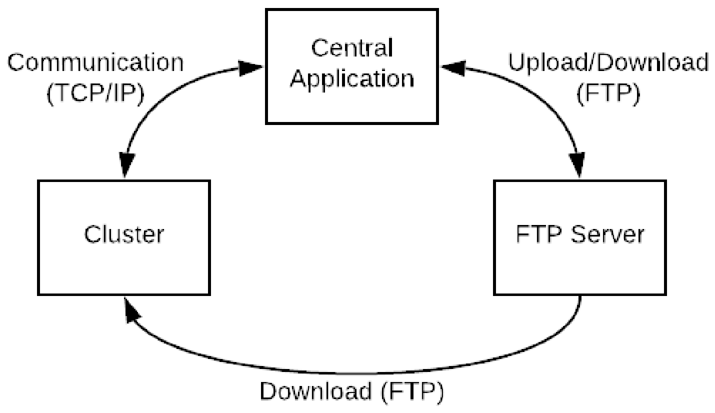


Figure 3.1: System Overview

3.5.1.1 System Flow

The purpose of the present section is to explain succinctly the flow and behavior of the system.

At first, the Central Application and the microcontrollers are not connected. One at a time, the microcontrollers will connect to the Central App, and thus become part of the cluster. From this point on, the information of each microcontroller will be displayed in the GUI.

At this point, the monitoring has already begun, as the status of the OS and connection of each microcontroller is known and displayed in the GUI. In case there is an error or OS change request, a message issuing the command to install a new OS and containing the path to the respective image in the FTP server is sent, and the installation begins.

The error detection is made through a periodic message that the Central Application sends to each microcontroller, which generates an echo message to verify that the microcontroller is still running, in case this echo message is not received on the Central Application it is assumed that there was an error and the command to reinstall the image is issued. The type or specific error is not important, because the objective is to solve it as fast and efficiently as possible to get the microcontroller running again.

3.5.2 Use Cases

This use case representation [3.2] contains all the interactions among the system's elements. Each of the elements will be analyzed individually.

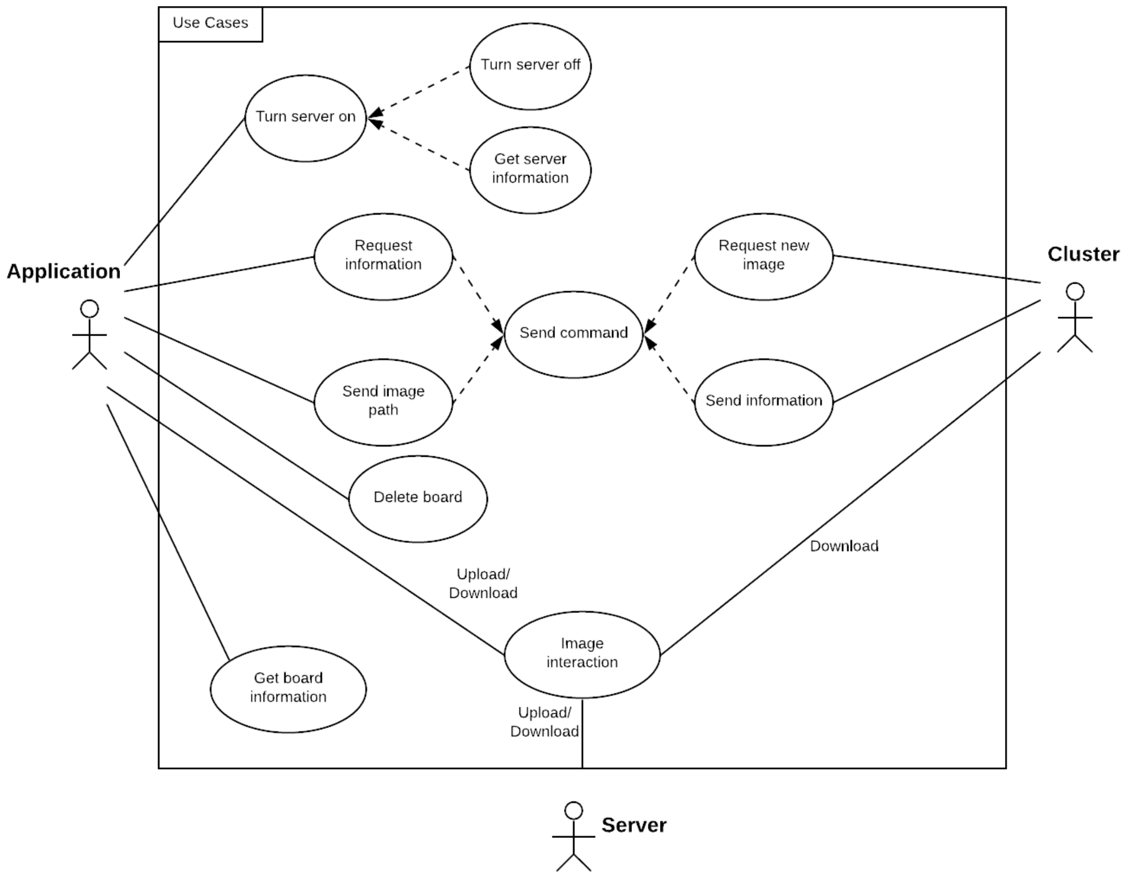


Figure 3.2: System Use Cases

Application

- Server - The Central Application is not predefined to accept connections from the cluster's Raspberry Pi, it must first be turned on. After that, information about it is displayed in the GUI, and it can be turned off at any time;
- Microcontroller management - The user has the option to remove a microcontroller from the server and to request the most recent information (OS, MAC and IP addresses)from it;
- Microcontroller control - Commands can be sent to every microcontroller, issuing a specific command, for example sending an image path for a new OS installation or request information about the booting and connection status of the microcontroller.

- FTP Server - OS images can be uploaded and downloaded from the FTP server

Server

- OS images - The server can download images (upon request) into the Central Application and receive images from it too. The server also transfers OS images into the cluster's Raspberry Pi upon request.

Cluster

- New OS - Each of the cluster's Raspberry Pi can autonomously choose to install a new OS, in this case it sends a request for the desired OS location in the FTP server to the Central Application;
- Response - Upon information request (either status or device specific information) the Raspberry Pi responds with the corresponding information.

Chapter 4

Implementation

4.1 Communication between Host and Client

4.1.1 Communication language

A set of commands has been defined, to simplify and limit the communication. Only messages that are in the predefined instruction set will be significant, otherwise they will be ignored. Considering the functionalities of the system, the commands in the table are essencial. The decision of chosing those commands was made having as parameters necessity and simplicity, having just what is necessary. One of the first aspects taken in consideration when creating the commands was the syntax, the command had to be easy to interpret by the receiver, whoever it might be. The solution was to use special characters to signal the beggining of the command and the beggining of relevant fields within the command. Another concern was that the commands were intuitive and self explanatory, which was easily achieved by basically naming each command their role.

To better understand the table below, the explanation of each field follows:

- **Command** - Specifies the name of the command
- **Sub-command** - Specifies the syntax that must follow the command name;
- **Examples** - Gives the example of a full command with the correct syntax and information (some commands do not have an example because they are consituted of their name only);
- **Direction** - Informs whether the command is sent from the RPi to the Central App or the other way around.

Command	Sub-command	Examples	Direction
#command		#command cd #command reboot	Central-RPi
#commandoutput		#commandoutput Shutting down in 10 sec	RPi-Central
#getstatus			Central-RPi
#status		#status Rebooting	RPi-Central
#newimage	$IP $USER $PW $PATH	#newimage $IP 10.42.0.4 $USER ftpuser $PW ftp $PATH /Raspbian	Central-RPi
#getinfo			Central-RPi
#info	$MAC $OS $ID	#info $MAC 00:0a:95:9d:68:16 $OS Raspbian $ID RPIn7	RPi-Central

Table 4.1: Command Table

Proceeding to the explanation of the purpose of each command:

- **command** - This command might seem a bit superfluous or useless, but it is in fact one of the most basic commands. It is the equivalent to having an opened terminal, it enables the user to send command line instructions;
- **commandoutput** - This command is bonded to the previous one, it contains the response of the machine who received the instruction specified in "#command", it is purely informative;
- **getstatus** - A server request on the booting stage/status of a specific microcontroller. This information will be used to keep the GUI updated;
- **status** - Response from the microcontroller to the previous command. Sends the current booting status of the microcontroller (Booting, Extracting Filesystem, ...);
- **newimage** - This command is of great importance, as it is the one where the Central App will send to any microcontroller the path of an image to be downloaded and burned. There is some more complexity to this command, as the download of the image from the FTP server. Given this, it is required to provide to the microcontroller the address of the server, the credentials (user and password) to gain access to the server, and the path of the image to be burnt;
- **getinfo** - This is a server request for information on a specific microcontroller, to update the GUI and facilitate user interaction and navigation. The information consists in the microcontroller's MAC address, current OS, and the microcontroller's desired designation;

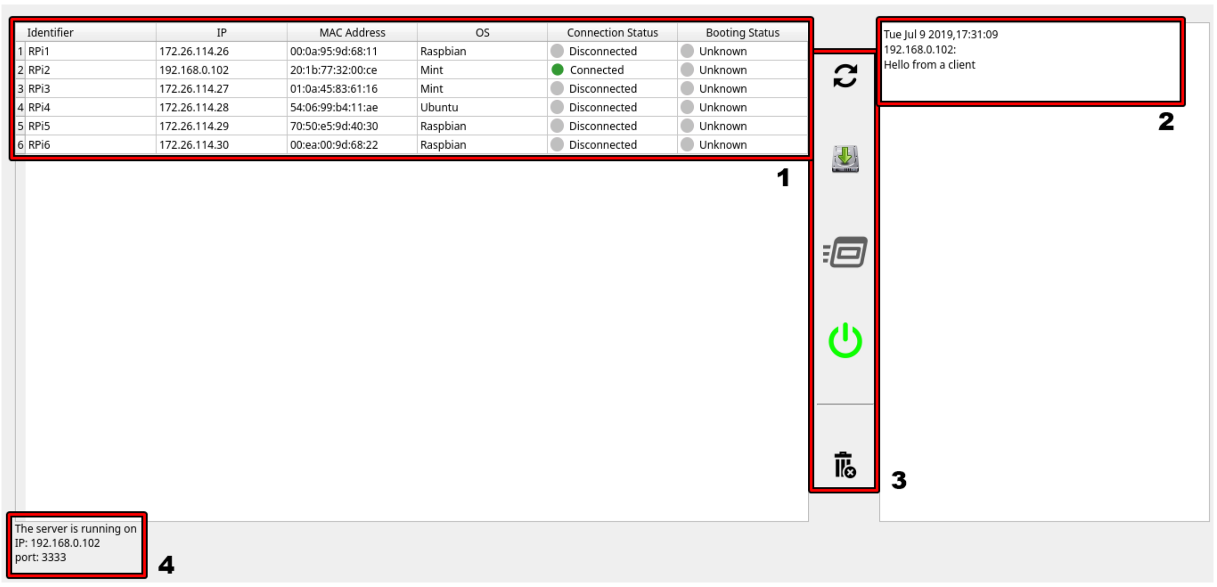
- **info** - Response to the previous command containing the before specified information.

As mentioned before, and clear after seeing the table, special characters were used to easy the parsing process, in this case the characters "#" and "$".

4.2 Graphical User Interface (GUI)

The Central Application obviously includes a Graphical User Interface, allowing the user to interact with each microcontroller and providing information about them. The GUI includes buttons, text display and input and intuitive icons.

4.2.1 Main Interface



	Identifier	IP	MAC Address	OS	Connection Status	Booting Status
1	RPi1	172.26.114.26	00:0a:95:9d:68:11	Raspbian	● Disconnected	● Unknown
2	RPi2	192.168.0.102	20:1b:77:32:00:ce	Mint	● Connected	● Unknown
3	RPi3	172.26.114.27	01:0a:45:83:61:16	Mint	● Disconnected	● Unknown
4	RPi4	172.26.114.28	54:06:99:b4:11:ae	Ubuntu	● Disconnected	● Unknown
5	RPi5	172.26.114.29	70:50:e5:9d:40:30	Raspbian	● Disconnected	● Unknown
6	RPi6	172.26.114.30	00:ea:00:9d:68:22	Raspbian	● Disconnected	● Unknown

Tue Jul 9 2019,17:31:09
192.168.0.102:
Hello from a client

The server is running on
IP: 192.168.0.102
port: 3333

Figure 4.1: Application's Main Graphical User Interface

The labels in the image above **[4.1]** represent the main elements of the GUI, by explaining each of them, the functioning of the GUI becomes clear:

1 - In this table is displayed the information about every microcontroller that is part of the network, all the information is provided by the microcontroller itself (sent to the Central App through TCP/IP):

- **Identifier** - Microcontroller designation chosen by the user;
- **IP** - Microcontroller's IP Address;

- **MAC Address** - Microcontroller's MAC Address;
- **OS** - Microcontroller's current OS;
- **Connection Status** - Current connection (to the Central App) state. The microcontroller can either be in "Connected", "Disconnected", or "Error" state, the last meaning there was some error during the connection attempt;
- **Booting Status** - Current image deployment and boot status. The microcontroller updates its image burning process status on the fly;

2 - This text box displays every received message, regardless of the microcontroller that sent it. For every message received and displayed, the current date and time are displayed as well. The IP of the microcontroller that sent the message is also printed on the screen;

3 - These are the five buttons which the user uses to interact and manage the microcontrollers in the network. From top to bottom, the buttons are:

- **Refresh Button** - Refresh the displayed information. The interface is automatically refreshed when necessary, but the user has the option to refresh it manually;
- **FTP Button** - Opens up a window to connect to an FTP server to upload or download files. This will be explained in more detail up ahead;
- **Command Button** - Opens up a window to send a command to the selected microcontrollers. This will be explained in more detail up ahead;
- **Power Button** - Brings the Central Application's server up and shuts it down;
- **Delete Board Button** - Used to remove a microcontroller from the network.

4 - In this field is the information about the current state of the server, if it is ON or OFF, and also includes the server IP and port address if it is on.

4.2.2 Delete Raspberry Interface

This is a very simple window, the user just selects which microcontroller to delete, from the dropbox, and then presses the button. The connection to the microcontroller will be closed and the microcontroller will be removed from the network.

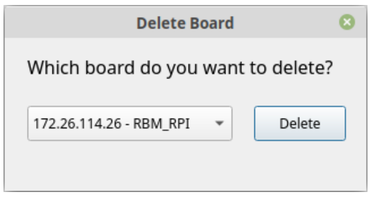


Figure 4.2: Delete Board Graphical Interface

4.2.3 Send Command Interface

The user must select at least one microcontroller to send a command to (but can choose multiple microcontrollers, even all of them if necessary) and then press the "Send Command" button, the list of commands is then provided in a dropbox. The commands are the ones listed on the table above **[4.1]**. All the commands are automatically sent with the correct formatation. After selecting the command and pressing ok, depending on the command, it will either be sent directly or the user will be asked to complement it. If it is a "#getinfo" or "#getstatus" command it is sent directly, if it is a "#command" type, then the terminal opens up and remains that way until the user types the command "exit". If the user choses the "#newimage" command, the FTP interface opens up, where the user , after connecting to the server (with correct username and password, if necessary), must choose a directory to send to the microcontroller. This last clase will be clearer in the next section.

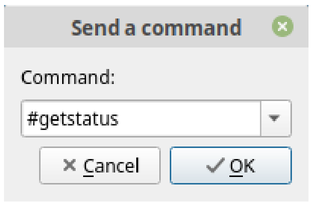


Figure 4.3: Send Command Graphical Interface

4.2.4 FTP Interface

The FTP interface is a bit more complex, as it has more needs than the previous ones. With the image below as support, the interface explanation follows:

- **1** - This section consists on the first step of the retrieving or upload of a file to the FTP server, the connection. The user must type the address of the server and then press the "connect" button. This connection will only be sucessful if the username and password (if necessary) are correct, which takes us to the second part. The small button next to the address box will be left for later;

- **2** - As mentioned, this part includes the username and password fields that have to be properly filled, or left empty, depending on the server security. In addiction to that there are also three buttons. The ”Upload” button opens up a dialogue so the user can choose a file to upload, the ”Download” button downloads a file from the server, but is only enabled once a file has been selected. The ”Send Path” button is only enabled when the ”#newimage” command is to be sent, as a part of this command is the path of the image to be downloaded by the microcontroller. This last button adds the path of the selected image to that command;
- **3** - This box displays all the files on the server’s current directory, once the connection to the server has been established. The user can navigate through the directories on te server (if the server so allows), and the small button on section 1 is used to navigate to the previous directory, if available. Once one of the displayed file has been selected, the ”Download” button becomes available and the user can download a file, the same goes if the user wants to add a path to the ”#newimage” command. The bar on the bottom shows the current percentage of progress of the download or upload of a file to the server.

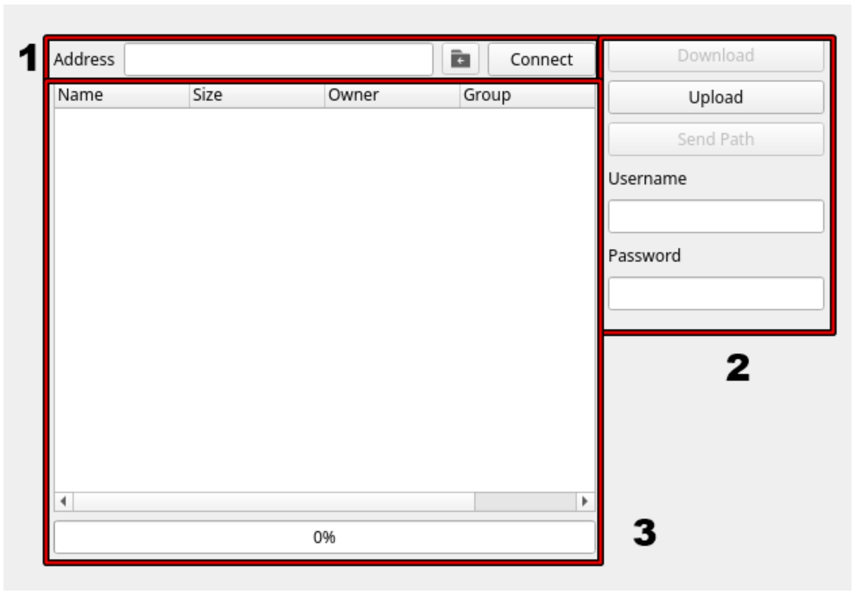


Figure 4.4: FTP Graphical Interface

The FTP server is a crucial part of the project, therefore the interface is designed to be as intuitive and practical as possible, discarding all that is not necessary and keeping only the essential elements to ensure its proper functioning.

Chapter 5

Tests and Results

5.1 Tests

The tests were ran using the Remote Boot Manager and a dummy client, hosted on the same computer (simulating the private LAN network), that connects and communicates with the Central Application. Some of the tests had to be done with a Raspberry Pi to test the burning of a new OS image. It is not regarded here, but of course the debugging tool was the quintessential test.

5.1.1 Turn Server On/OFF

The Central Application server is not receptive to connections when the program is run, it has to be manually turned on.

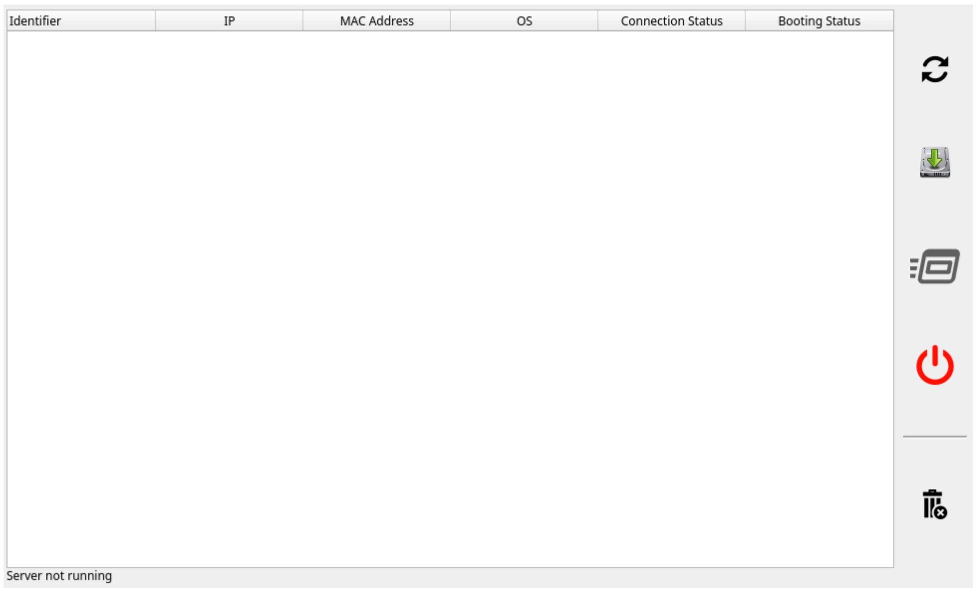
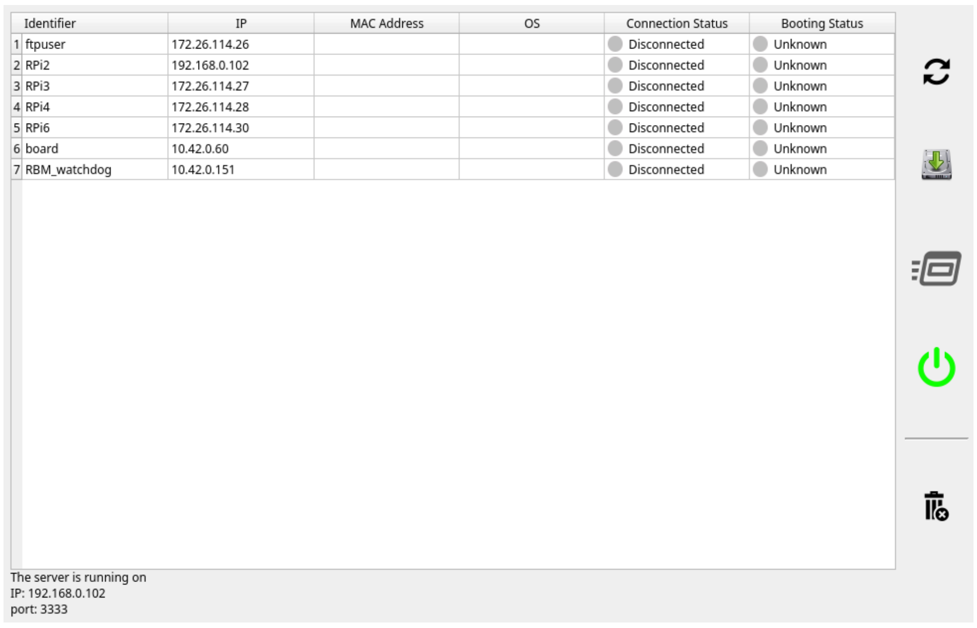


Figure 5.1: Turn Server On Test (1)



	Identifier	IP	MAC Address	OS	Connection Status	Booting Status
1	ftpuser	172.26.114.26			● Disconnected	● Unknown
2	RPi2	192.168.0.102			● Disconnected	● Unknown
3	RPi3	172.26.114.27			● Disconnected	● Unknown
4	RPi4	172.26.114.28			● Disconnected	● Unknown
5	RPi6	172.26.114.30			● Disconnected	● Unknown
6	board	10.42.0.60			● Disconnected	● Unknown
7	RBM_watchdog	10.42.0.151			● Disconnected	● Unknown

The server is running on
IP: 192.168.0.102
port: 3333

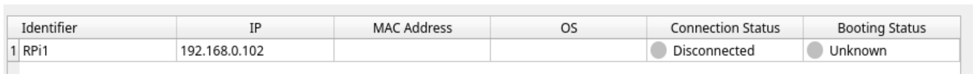
Figure 5.2: Turn Server On Test (2)

When the server is turned off, the power icon on the GUI appears red and on the bottom of the widget there is a message saying ”Server not running” [5.1]. Once the server is turned on, by pressing the power icon, the same icon turns green and the information (IP and port) about the server is displayed [5.2]. Also, once it is turned on, the Raspberry Pi that are part of the cluster are displayed on the widget [5.2].

This process allows for a detection of the pressing of the button and if the server is properly turned on, as the information would not appear if it had not been so.

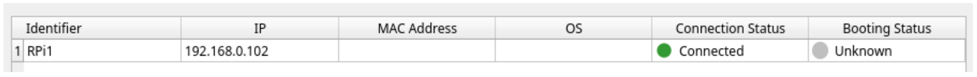
5.1.2 Connect Board

In the Raspberry Pi display widget, if a microcontroller is disconnected, the Connection Status is matching, and the circle is gray [5.3]. Once the microcontroller connects, the GUI is immediately and automatically updated and the state changes to ”Connected” and the circle turns green [5.3].



	Identifier	IP	MAC Address	OS	Connection Status	Booting Status
1	RPi1	192.168.0.102			● Disconnected	● Unknown

Figure 5.3: Connect Board Test (1)



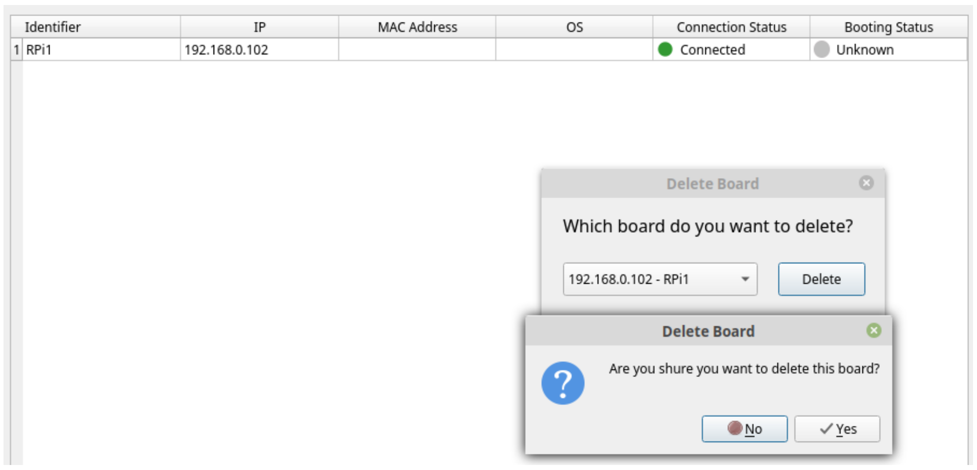
	Identifier	IP	MAC Address	OS	Connection Status	Booting Status
1	RPi1	192.168.0.102			🟢 Connected	● Unknown

Figure 5.4: Connect Board Test (2)

In early tests the status updating process occurred manually, it was a faster and simpler way of testing. It then evolved to a periodic update, until the last and most efficient method was implemented, a signal from the connection socket that was triggered every time the connection state changes.

5.1.3 Remove Board

To remove a microcontroller, the button in the GUI is pressed, after that there is a dropbox for selecting the microcontroller to be deleted. Once selected, there is a confirmation message for the deletion of the microcontroller. After confirming, the connection to the microcontroller is terminated, the microcontroller is deleted from the database and therefore from the GUI.



	Identifier	IP	MAC Address	OS	Connection Status	Booting Status
1	RPi1	192.168.0.102			● Connected	● Unknown

Delete Board

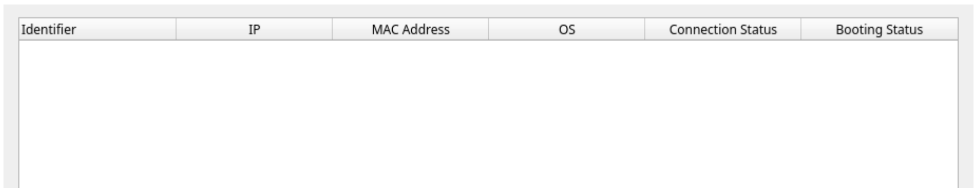
Which board do you want to delete?

192.168.0.102 - RPi1

Delete Board

Are you shure you want to delete this board?

Figure 5.5: Delete Board Test (1)

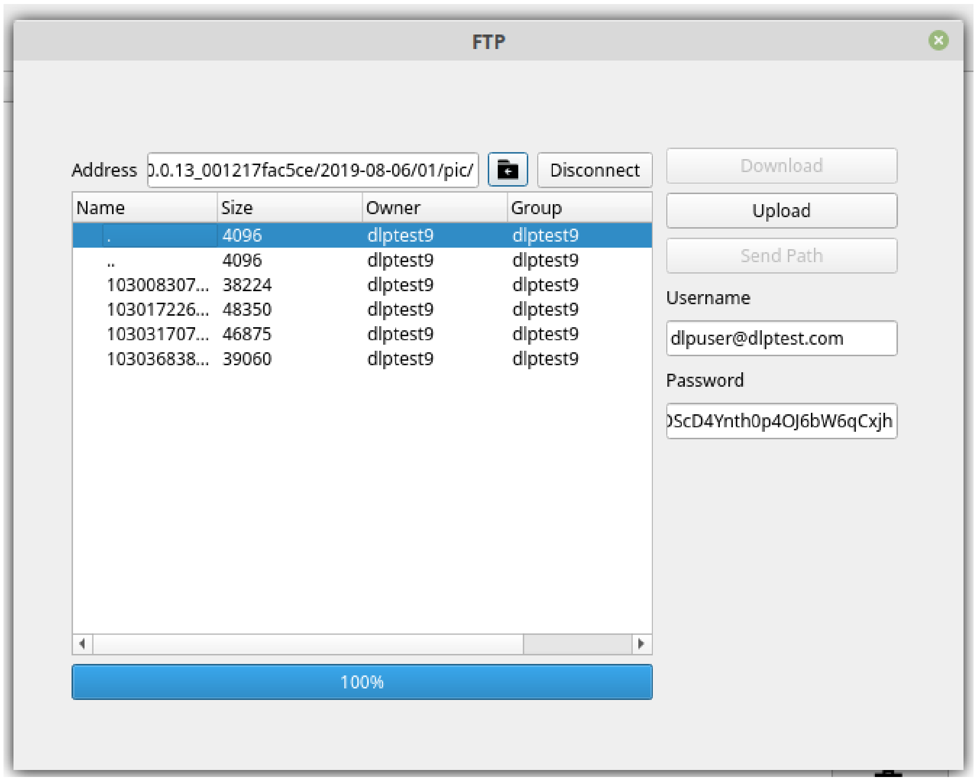


Identifier	IP	MAC Address	OS	Connection Status	Booting Status

Figure 5.6: Delete Board Test (2)

5.1.4 Upload File

In order to test the upload of a file a public FTP server was used. This test includes not only the upload of a file but also the connection to the FTP server. Due to its importance and higher quantity of steps, the testing of this part os the project took longer.



Address:).0.13_001217fac5ce/2019-08-06/01/pic/

Name	Size	Owner	Group
.	4096	dlptest9	dlptest9
..	4096	dlptest9	dlptest9
103008307...	38224	dlptest9	dlptest9
103017226...	48350	dlptest9	dlptest9
103031707...	46875	dlptest9	dlptest9
103036838...	39060	dlptest9	dlptest9

Username: dlpuser@dlptest.com

Password:)ScD4Ynth0p4OJ6bW6qCxjh

Figure 5.7: Upload File Test (1)

The first stage consists of clicking the FTP button on the GUI to bring up the FTP dialog. Once it has popped up, it is necessary to input the server's address and credentials (username and password) and press the "Connect" button. Having completed these steps, the files on the server will be displayed on the dialog [5.7], this means the connection has been successfully established.

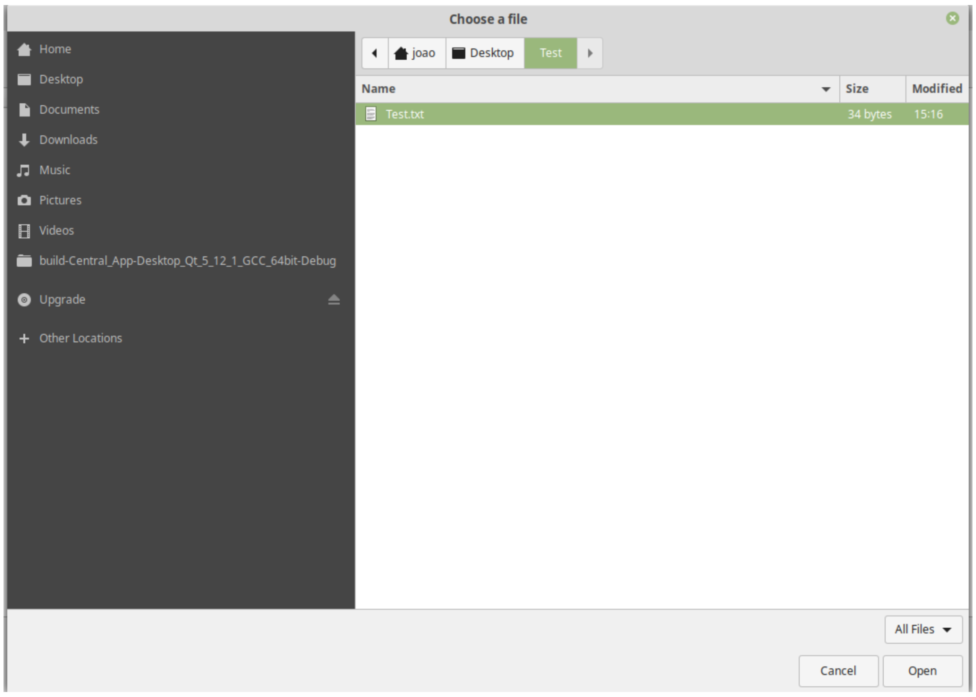
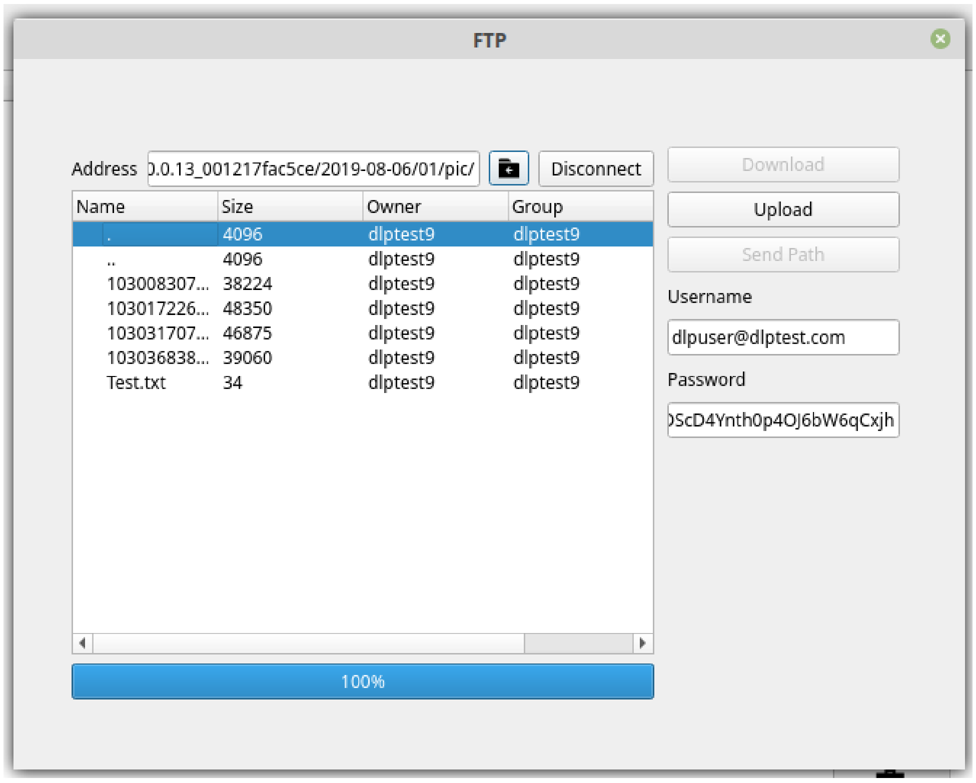


Figure 5.8: Upload File Test (2)

The next step is to click on "Upload" on the same window [5.7]. This will bring up another window to select the file to be uploaded [5.8].



Address:).0.13_001217fac5ce/2019-08-06/01/pic/

Name	Size	Owner	Group
.	4096	dlptest9	dlptest9
..	4096	dlptest9	dlptest9
103008307...	38224	dlptest9	dlptest9
103017226...	48350	dlptest9	dlptest9
103031707...	46875	dlptest9	dlptest9
103036838...	39060	dlptest9	dlptest9
Test.txt	34	dlptest9	dlptest9

Username: dlpuser@dlptest.com

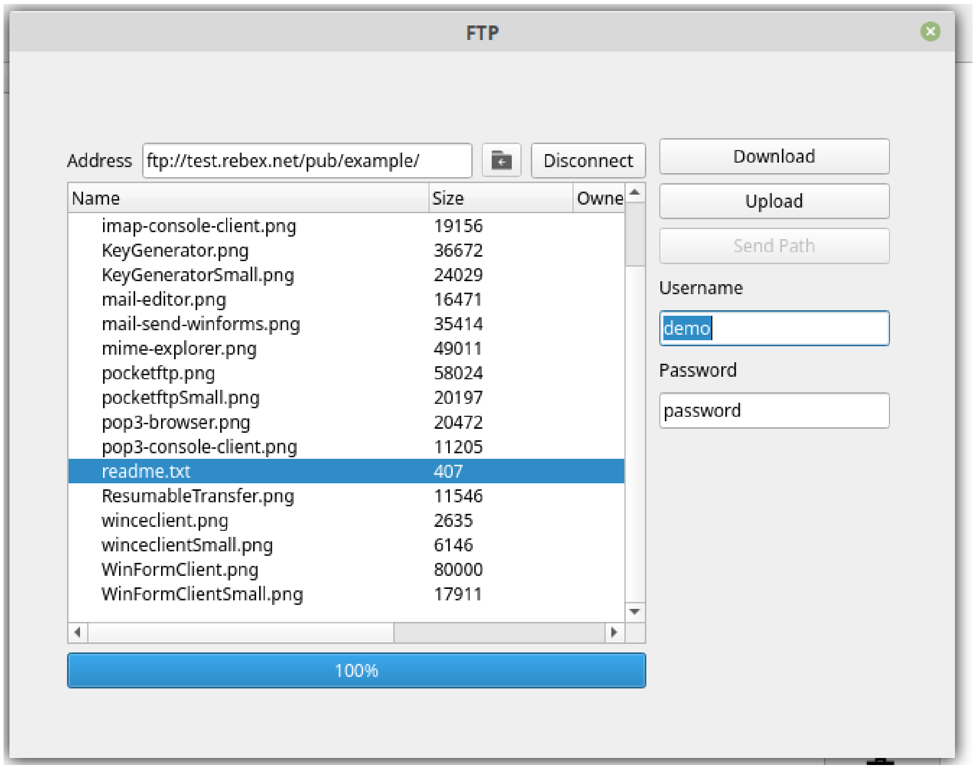
Password:)ScD4Ynth0p4OJ6bW6qCxjh

Figure 5.9: Upload File Test (3)

After selecting the file, it will start being uploaded, the progress bar indicates the percentage of progress of the process. Once the upload is finished, the file will appear in the server's file list [5.9].

5.1.5 Download File

The testing of the Downloading of a file was very similar to the testing of the Upload. The process starts pressing the FTP button on the GUI and then inputting the server's address and the credentials for having access [5.10].



Address ftp://test.rebex.net/pub/example/

Name	Size	Owne
imap-console-client.png	19156	
KeyGenerator.png	36672	
KeyGeneratorSmall.png	24029	
mail-editor.png	16471	
mail-send-winforms.png	35414	
mime-explorer.png	49011	
pocketftp.png	58024	
pocketftpSmall.png	20197	
pop3-browser.png	20472	
pop3-console-client.png	11205	
readme.txt	407	
ResumableTransfer.png	11546	
winceclient.png	2635	
winceclientSmall.png	6146	
WinFormClient.png	80000	
WinFormClientSmall.png	17911	

Username demo

Password password

Figure 5.10: Download File Test (1)

The next step is to browse through the available files displayed and select the one to be downloaded. Having selected the file, press the "Download" button [5.10]. This part of the testing is very important, because the file path has to be retrieved right, otherwise the file will not be downloaded.

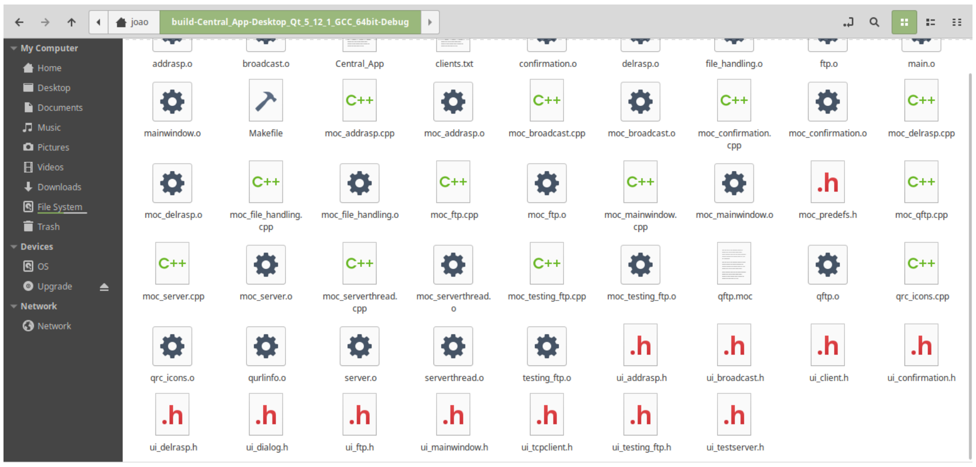


Figure 5.11: Download File Test (2)

The progress bar once again gives the percentage of progress of the process [5.10]. Once the process is completed, a popup window will display a success message saying the file has been downloaded.

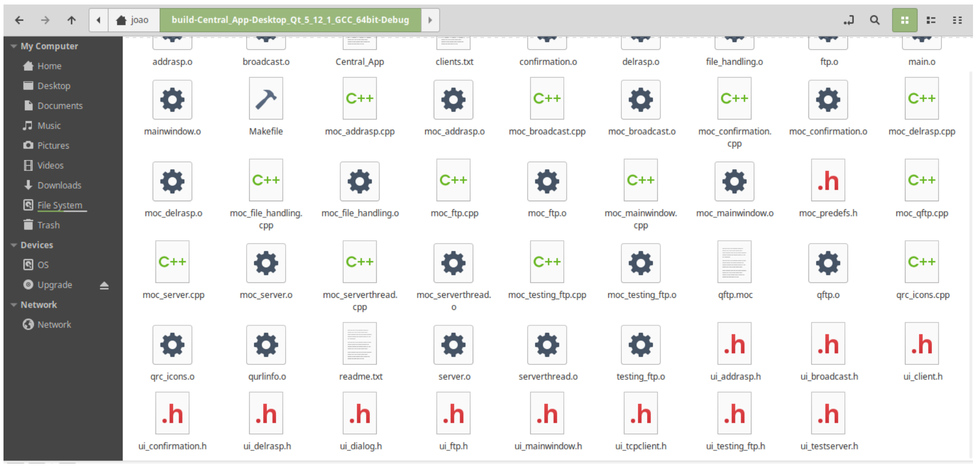
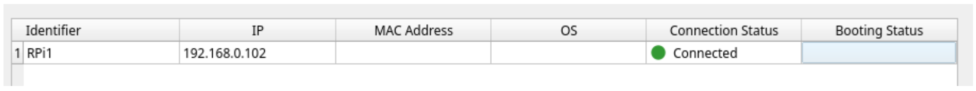


Figure 5.12: Download File Test (3)

Upon going to the project's folder, it is possible to see the file that previously was not there [5.11], is now in the folder, meaning it has successfully been downloaded.

5.1.6 Board Response

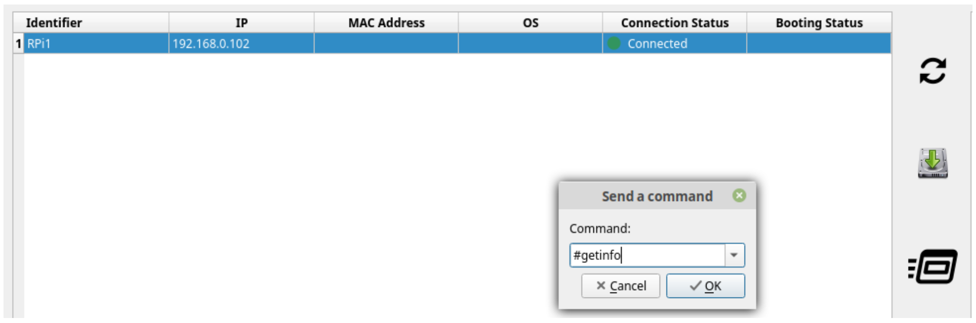
This test includes two types of response. One of them is a reply with information [5.15], the second is reflected in actions [5.16]. Both of them come from an initial request from the Central Application.



	Identifier	IP	MAC Address	OS	Connection Status	Booting Status
1	RPi1	192.168.0.102			● Connected	

Figure 5.13: Raspberry Pi Response Test (1)

The initial state of the GUI is visible in the first figure [5.13]. The test is conducted by starting at this point and then issuing a command to the microcontroller and seeing how the GUI changed.



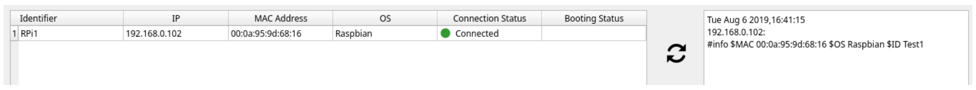
	Identifier	IP	MAC Address	OS	Connection Status	Booting Status
1	RPi1	192.168.0.102			● Connected	

Send a command

Command: #getinfo

Figure 5.14: Raspberry Pi Response Test (2)

First, the microcontroller to issue the command to has to be selected, then the "Send Command" button in the GUI has to be pressed. A window will pop up, and the command can be selected from a dropbox [5.14].



	Identifier	IP	MAC Address	OS	Connection Status	Booting Status
1	RPi1	192.168.0.102	00:0a:95:9d:68:16	Raspbian	● Connected	

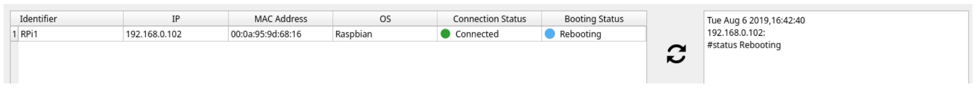
```
Tue Aug 6 2019,16:41:15
192.168.0.102:
#info $MAC 00:0a:95:9d:68:16 $OS Raspbian $ID Test1
```

Figure 5.15: Raspberry Pi Response Test (3)

The first test consists on requesting a microcontroller information (MAC and IP addresses, current OS) on it. The command to make this request is "#getinfo".

The result is on the image above [5.15]. On the right is the raw message received from the microcontroller the request has been sent to, this message includes a header with the date and time the message was received, and the IP of the microcontroller who sent it. The contents of the message are the ones mentioned before. The desired microcontroller designation in this cased was received too, this is optional, as each microcontroller is assigned a predefined designation, but if the user wants to assign a specific name to a microcontroller, then that name can be sent too.

Besides the received message, and more important, the received information is now displayed on the respective fields, in the row of the microcontroller who sent it [5.15].



	Identifier	IP	MAC Address	OS	Connection Status	Booting Status
1	RPi1	192.168.0.102	00:0a:95:9d:68:16	Raspbian	● Connected	● Rebooting

```
Tue Aug 6 2019,16:42:40
192.168.0.102:
#status Rebooting
```

Figure 5.16: Raspberry Pi Response Test (4)

The last test begins the same way, by sending a command [5.14], but this time the command is ”#newimage”. This command issues the installing of a new OS on the target microcontroller and it includes the path to that OS image.

The microcontroller replies with its new status, as seen on the received message on the right [5.16], and the Booting Status is updated on the respective microcontroller row, and the circle turns blue [5.16], indicating it is a temporary state (as the booting is comprised of many states, changing rapidly).

This testing is very important because the coherence of the GUI and making sure it always has updated information is very important for the managing of the cluster by the user.

5.2 Tests Results

The tests that were ran were the ones needed to test the last stages of a process and their ultimate purpose. Most of the intermediary test were made using the debugger, as in every programming process, and those are not contemplated here.

The tests were an important and significant part of the developing process of the system, to make sure that that process was being done properly. As Qt provides an easy implementation of a GUI, it became easy to run continuous graphic tests while programming.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This project aimed to facilitate and to reduce needed resources for cluster management. Based on the research that had been previously done, it was concluded that this was possible, by integrating and assembling various technologies and approaches. The results coming from the practical implementation of this hypothesis proved that the monitoring of the cluster's Raspberry Pi becomes way easier and the time spent is largely reduced.

All of the proposed objectives were met, some of them took longer than expected, not only but also due to some limitations, for example the lack of a closed local network to conduct the necessary tests.

Having achieved granting the display of updated information on the cluster at all times and being able to communicate simultaneously with all the microcontrollers that are part of it was a major and fundamental accomplishment for the fulfillment of the project.

The research and implementation process illustrates that time and effort while monitoring a cluster could be reduced, but it also demonstrates that the integration of different ideas and technologies can bring up numerous advantages. Further research on different approaches/methods to this issue might take the improvements even further, as has been found to have been done in previous projects [5]. The application developed shows a useful and intuitive way of managing clusters and poses as a good alternative with unique features lighten this task.

6.2 Future Work

Further research is needed in the security aspect. This Thesis did not take into account security concerns, therefore no security measures were taken. But in systems that include communications and networks, security is a very important asset. Looking forward, the development of security would greatly improve and add value to the developed project, as well as more credibility and visibility.

Another topic to do further research on is to broaden the microcontroller cluster type. The Raspberry Pi is frequently used, but many clusters already in use, use other type of microcontrollers. With this in mind, regarding future developments, this is definitely an important one, as it would make the Boot Manager much more versatile.

An important issue to solve is the case when a recurrent failure or error happens. At present, if this occurs, the image will continuously be reinstalled on the microcontroller, which is ineffective and useless. Developing a measure to counter this issue would add value to the system.

References

[1] the free encyclopedia Wikipedia. *Computer cluster*. July 8, 2019. URL: `https://en.wikipedia.org/wiki/Computer_cluster`.

[2] Albert Dias. *The making of a microcontroller cluster*. May 1, 2003. URL: `https://albertdias.blog/2003/05/01/microcontroller-cluster/`.

[3] techopedia. *Clustering*. URL: `https://www.techopedia.com/definition/17/clustering-databases`.

[4] Cypress. *Microcontrollers for Instrument Clusters*. URL: `https://www.cypress.com/applications/automotive-solutions/microcontrollers-instrument-clusters`.

[5] I. Firmansyah ; Z. Akbar ; L.T. Handoko. “Microcontroller based distributed and networked control system for public cluster”. In: *Group for Theoretical and Computational Physics, Research Center for Physics, Indonesian Institute of Sciences (LIPI)* (). URL: `https://arxiv.org/pdf/0906.0281.pdf`.

[6] Tejas Ambekar Tejas Ambekar. *What is cluster computing? List disadvantages of cluster computing?* July 9, 2018. URL: `https://www.quora.com/What-is-cluster-computing-List-disadvantages-of-cluster-computing`.

[7] Vladimir V. Mazur ; Karine A. Barmuta ; Sergey S. Demin ; Evgeny A. Tikhomirov ; Maxsim A. Bykovskiy. “Innovation Clusters: Advantages and Disadvantages”. In: *International Journal of Economics and Financial Issues* (2016). URL: `https://dergipark.org.tr/download/article-file/363503`.

[8] I. Ahmed ; Hong Wong ; V. Kapila. “Internet-based remote control using a microcontroller and an embedded Ethernet”. In: *IEEE Xplore Digital Library* (Jan. 24, 2005). URL: `https://ieeexplore.ieee.org/document/1386759`.

[9] Nor Azlan Othman ; Muhammad Riduan Zainodin ; Norhasnelly Anuar ; Nor Salwa Damanhuri. "Remote monitoring system development via Raspberry-Pi for small scale standalone PV plant". In: *IEEE Xplore Digital Library* (Feb. 8, 2018). URL: `https://ieeexplore.ieee.org/document/8284435`.

[10] Purnomo Husnul Khotimah ; Dikdik Krisnandi ; Bambang Sugiarto. "Design and implementation of Remote Terminal Unit on Mini Monitoring Weather Station Based on Microcontroller". In: *IEEE Xplore Digital Library* (Dec. 8, 2011). URL: `https://ieeexplore.ieee.org/document/6095431`.

[11] Sebastian Schmelzer ; Dirk von Suchodoletz ; Gerhard Schneider ; Daniel Weingaertner ; Luis Carlos E. de Bona ; Carlos Carvalho. "Universal remote boot and administration service". In: *IEEE Xplore Digital Library* (Dec. 15, 2011). URL: `https://ieeexplore.ieee.org/document/6102270`.

[12] Dataplicity. *It's the foundation for IoT remote support*. 2019. URL: `https://www.dataplicity.com/features/`.

[13] Tosibox. *Build your IoT ecosystem with TOSIBOX®*. 2019. URL: `https://www.tosibox.com/tosibox-iot-ecosystem/`.

[14] Balena. *What is balena?* 2019. URL: `https://www.balena.io/what-is-balena`.

[15] the free encyclopedia Wikipedia. *Internet protocol suite*. Aug. 4, 2019. URL: `https://en.wikipedia.org/wiki/Internet_protocol_suite`.

[16] a enciclopédia livre Wikipédia. *Servidor FTP*. June 30, 2019. URL: `https://pt.wikipedia.org/wiki/Servidor_FTP`.

[17] the free encyclopedia Wikipedia. *File Transfer Protocol*. Aug. 3, 2019. URL: `https://en.wikipedia.org/wiki/File_Transfer_Protocol`.

[18] Rembo Technology SaRL. *An introduction to remote-boot*. URL: `http://www.bpbatch.org/docs/remoteboot.html`.

[19] the free encyclopedia Wikipedia. *Network booting*. July 27, 2019. URL: `https://en.wikipedia.org/wiki/Network_booting`.

[20] Carlos Neves; José Matos; Luis Neves. *SUSTIMS - PLATAFORMA DE GESTÃO SUSTENTÁVEL DE INFRAESTRUTURAS RODOVIÁRIAS*. URL: `http://www.crp.pt/docs/A48S186-8_CRP_T5_074.pdf`.

[21] Luís Miguel Martins Pereira. "Sistema ciber-físico para deteção de colisões". type. Universidade do Minho, 2019.

[22] Sustain. *Why bees are important*. URL: https://www.sustainweb.org/foodfacts/bees_are_important/.

[23] the free encyclopedia Wikipedia. *Beehive*. Sept. 13, 2019. URL: https://en.wikipedia.org/wiki/Beehive#Traditional_hives.

[24] Hilary. *10 MISTAKES NEW BEEKEEPERS MAKE*. Dec. 7, 2015. URL: https://beekeepinglikeagirl.com/10-mistakes-new-beekeepers-make/.

[25] Ulf Eriksson. *Functional vs Non Functional Requirements*. Apr. 5, 2012. URL: https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/.

[26] the free encyclopedia Wikipedia. *Raspberry Pi*. Aug. 3, 2019. URL: https://en.wikipedia.org/wiki/Raspberry_Pi.

[27] Raspberry Pi Foundation. *Raspberry Pi 3 Model B+*. URL: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/.

[28] the free encyclopedia Wikipedia. *Qt (software)*. July 30, 2019. URL: https://en.wikipedia.org/wiki/Qt_(software).

[29] Qt. *What do you get with Qt?* 2019. URL: https://www.qt.io/what-is-qt/?utm_campaign=Navigation%202019&utm_source=megamenu.

[30] the free encyclopedia Wikipedia. *Constraint (mathematics)*. May 18, 2019. URL: https://en.wikipedia.org/wiki/Constraint_(mathematics).

[31] Institut für Informatik. *Lehrstuhl für Informatik VIII - Informationstechnik für Luft- und Raumfahrt*. URL: http://www8.informatik.uni-wuerzburg.de/wissenschaftforschung/honeycloud/.