Technical University of Denmark

# Modular Playware Embedded System

Bachelor's Thesis


Author

**Manuel Gil Martínez**

Supervisor

**Henrik Hautop Lund**



DTU Electrical Engineering

*Center of Playware*

Danmarks Tekniske Universitet

Copenhagen, Denmark 2017

Modular Playware Embedded System
Manuel Gil Martínez

Supervisor: Prof. Henrik Hautop Lund, DTU Electrical Engineering
Co-Supervisor: Massimiliano Leggieri

Bachelor's Thesis 2017
Department of Electrical Engineering
Center of Playware
Danmarks Tekniske Universitet
DK-2800 Kgs. Lyngby, Copenhagen

Typeset in LaTeX
Copenhagen, Denmark 2017

# Acknowledgements

Special thanks to my Supervisors Henrik Hautop Lund and Massimiliano Leggieri, for their guidance through this Thesis Project, for solving my doubts and for giving me the opportunity to discover and to learn about Playware Technology. Many thanks to my family for their support, and last but not least, to the DTU which has provided me with a great learning environment and has been part of this enriching experience during the last five months.

Manuel Gil Martínez, Copenhagen, June 2017

# Summary

This project has the objective of developing modular playware units that act as interactives nodes in a network. The control of the modules will be divided between on-board control on the embedded system modules and control from a network connected smart device (e.g. Android tablet or smartphone). This project also examines the division of control between the individual modules and the smart device, and the best protocol for communication between them. The focus is working in low level software for controlling the embedded systems, more specifically using the ESP chip which provides WiFi connectivity between devices as well as between devices and the Android device. The project also investigates the on-board sensors and actuators of the boards with the objective of providing playware experiences.

Taking all of this into account, and after testing and researching about all these different topics, some real-time demonstrations have been created to expand all the capabilities and they will be exposed and bring up to discussion at the time of oral presentation.

# Contents

# List of Acronyms

| | | |
|---|---|---|
| **ADC** | — | Analog-to-Digital Converter |
| **API** | — | Application Programming Interface |
| **AP** | — | Access Point |
| **EEPROM** | — | Electrically Erasable Programmable Read-Only Memory |
| **FTDI** | — | Future Technology Devices International |
| **FTP** | — | File Transfer Protocol |
| **GPIO** | — | General-Purpose Input/Output |
| **HTTP** | — | Hypertext Transfer Protocol |
| **I2C/IIC** | — | Inter-Integrated Circuit |
| **I2S** | — | Inter-IC Sound |
| **IDE** | — | Integrated Development Environment |
| **ID** | — | Identifier |
| **IP** | — | Internet Protocol |
| **IoT** | — | Internet of Things |
| **LED** | — | Light-Emitting Diode |
| **MCU** | — | Microcontroller Unit |
| **MC** | — | Modular Cube |
| **MIDI** | — | Musical Instrument Digital Interface |
| **MQTT** | — | Message Queue Telemetry Transport |
| **OS** | — | Operating System |
| **OTA** | — | Over The Air |
| **PCB** | — | Printed Circuit Board |
| **PC** | — | Personal Computer |
| **PWM** | — | Pulse Width Modulation |
| **SCL** | — | Serial Clock |
| **SDA** | — | Serial Data Signal |
| **SDK** | — | Software Development Kit |
| **SPI** | — | Serial Peripheral Interface |
| **STA** | — | Station |
| **TCP** | — | Transmission Control Protocol |
| **UART** | — | Universal Asynchronous Receiver/Transmitter |
| **UDP** | — | User Datagram Protocol |
| **USB** | — | Universal Serial Bus |
| **WAP** | — | WiFi Protected Access |
| **WPS** | — | WiFi Protected Setup |
| **WiFi** | — | Wireless Fidelity |
| **iOS** | — | iPhone Operating System |

# 1
# Introduction

## 1.1 Objectives and general methodologies

The subject of this thesis is the creation of a network of modular devices that can interact between them in realtime to create a playful experience for the users. The main requirement given by the supervisors to achieve this goal was to use the ESP8266 WiFi chips. Wireless electronics are in our everyday life, and the growth of the Internet of Things has opened a new world of possibilities by using protocols that have been around us for a while, such as TCP/IP. The abovementioned chips have become the most inexpensive and most accessible choice for this kind of projects involving WiFi communications.

Hence the following questions arose:

- Which is the best protocol to send and receive messages between wireless embedded devices?
- How can these communications between modular devices be used to create a playware experience by interacting with the users?

Over the next chapters, an analysis of the best available options for a modular playware system is made and discussed. Once my choices are explained, there is an implementation of the system with the Tetra boards provided by the supervisor.

## 1.2 Motivation

The idea of providing playful experiences has always been in my mind since mid 2012, when I watched a documentary about how games were made and designed. That documentary motivated me to expand my knowledge in programming and to use the limited knowledge I had back then to start making games. Over the years, I have devoted some of my free time to develop video games as a hobby and at some point I decided to start my own Android game development company back in Spain. When I came to Denmark to do my Bachelor's Thesis and finish my Electronics and Automation degree I found out about the Center of Playware and the research being made in the field, mixing technology with play captivated me since the first minute. Combining my background in Electronics, my familiarity with programming gaming experiences and my passion in coding were some of the things that motivated me to contact Professor Lund in order to have my project supervised by him. The concept

behind this thesis was provided by both Henrik Hautop Lund, Professor of Robotics at DTU, and the Co-Supervisor Massimiliano Leggieri.

The motivation about the intriguing concept of playware technology has increased during the development of this Thesis Project. Technology can be a powerful instrument to improve people's life, and this aim is behind the research and development being carried out by the Center of Playware.

## 1.3 Playware Potential

One of the main purposes of this thesis is to broaden the possibilities of products and projects being developed in the Center of Playware. At the same time, it is also important to demostrate that the research carried out in this project about ESP8266 chips can enhance some of the already existing modular playware devices, such as the MOTO Tiles [12] or Fable, the socially interactive modular robot [16].

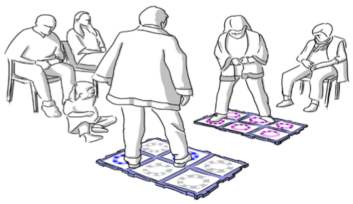

**Figure 1.1:** The MOTO tiles [12].

When compared with Bluetooth technology, WiFi communications expand the number of devices that can be connected simultaneously, including both smart and embedded system devices. Some fallouts can be experienced when connecting 6 or 7 devices via Bluetooth, something that does not happen with WiFi, where those limits are extended. On the other hand, WiFi is also a more common technology, included in the devices we use in our daily lives (i.e. Personal Computers, smartphones, tablets) and can be more convenient than for example ANT+, used in the MOTO Tiles. ANT+, despite of being fast and reliable, is not available in all the smart devices in the market.

This system based on WiFi can also open the doors to products that are directly connected to the internet, since each module could update itself or save information into a database.

For these reasons, there are endless potential applications for this modular playware system and some examples could be:

- Modular educational toys: A group of cubes with different numbers, shapes, words, animals or colors where the player, a child, has to select the correct option given by the Android/iOS device.

- Rehabilitation balls: Bearing in mind the idea of the interactive tiles, modular electronic balls that would help elder people to exercise their upper part of the body by moving it in different kinds of games. These balls could track their own movement and work in pairs of two, one for each hand. Different players at the same time would also make the games more interactive.

## 1.4 Report Structure

- Chapter 2 explains what a Modular Playware Embedded System is by defining each of the terms in the Thesis title.

- In Chapter 3, different network topologies are studied and discussed.

- Chapter 4 goes into detail on the network protocols tested during the development of the Thesis Project. The main reasons for choosing one among the others are explained in the last section, also including the time measurements of each protocol.

- In Chapter 5 and Chapter 6, the implementation of the project divided between Hardware and Software is explained. The information obtained during the testing process is explained in-depth as well as some of the most important code and thoughts behind the system.

- Finally, in Chapter 7 the conclusions and the possible future projects will be analyzed.

# 2

# What is a Modular Playware Embedded System?

This chapter aims to explain what a "Modular Playware Embedded System" is, by splitting the name into different sections and defining each of them in depth. The objective is to provide the reader with a clear view of the basic concepts behind this project by the end of this chapter.

## 2.1 Modular

Modular systems, also called self-configurable systems, are a series of autonomous elements with variable morphology. The modules are each of the elements that make up the whole modular system. Beyond a conventional robotic system, a modular system has to be able to self-configure, change its shape, and adapt to the changes of the different network elements regardless of its connection, disconnection or the information that may be received from the outside of the system.

Modular robotics have been around for a long time, but the accelerated development of network technologies, along with the lower costs and facility to produce electronics nowadays have made them more accessible, robust and low-cost than ever. A clear example to illustrate this is the Arduino project [18] used to develop this project, which makes building a prototype or even a final product faster, thanks to its reliability, online community and documentation.

Along this project, it has been assumed that modules are autonomous embedded systems. Each of the elements is able to act individually until it detects the presence of another possible node of the system. A list of assumptions and requirements was drafted for the design of this system. For example, the number of nodes that make up the system has to be indefinite and only limited by the hardware restrictions, such as the memory heap that can be allocated in a single element.

## 2.2 Playware

This term was first introduced in the paper *Playware - Intelligent technology for children's play* by Carsten Jessen and Henrik Hautop Lund, where is described as:

> *"intelligent hardware and software that creates play and playful experiences for users of all ages."* [11]

The research field of playware focuses into investigating the use of technology to create what is usually labeled as "play". Even if computer games are the most known type of playware, the term goes into fields such as modularity, interaction, robotics and AI among others. Lund and Jessen created the Center for Playware in 2009 at the Technical University of Denmark to research in those areas as well as to create playware products [13]. As a result of the research and development in the Center of Playware some products have been created in the area of exercise, creativity, rehabilitation, art, learning and innovation. One example of this will the MOTO tiles [12] that will be mentioned along this project.

## 2.3 Embedded System

An embedded system is a computer system designed thinking in a specific function, usually related to real-time computing. These devices are often based on microcontrollers and microprocessors, and since they are designed to work in a dedicated task, size, reliability and performance are the key features in their fabrication. We can find embedded systems in our day-to-day lives, from oven and fridges to audio players or plane control devices. The main microprocessor includes input and output interfaces and usually controls different actuators and sensors depending on the purpose of the system. In some cases an embedded system is just one of the pieces of a more complicated system with other electronic devices or mechanical parts.

Embedded systems can be programmed directly in the assembler language of the built-in microprocessor or microcontroller, but C and C++ are gradually becoming the most common coding language for this type of devices. There are of course some other options to be used, such as JAVA or Python wrappers. These can be helpful when testing but tend to be less efficient than working directly with the low-level code. Therefore, for the purpose of an embedded system, these languages are generally discarded in final products.

Among embedded systems, user interfaces are commonly customized for each application need, or even non-existent in some cases. The main reason of using embedded systems is the low production costs and the efficiency of a device that was designed with the final application in mind. Their possibilities are almost endless. Chapter 5 contains extended information about embedded systems and their components, going into detail on the two different boards used to test and make this project.

# 3

# Network Topology

Network topology is defined as the arrangement of the different nodes that build the network [8]. The layout that is used to connect the different elements of the network is called physical topology. In the same way, the path that the signals follow between the different nodes is a logical topology, and in some systems it can be different from the physical one [9].

This section will focus on the various topologies studied in order to find out the best one for this project, and will subsequently state the reasons taken into account for that purpose. Nevertheless, there are more topologies that were not considered for this thesis because of their limitations.

## 3.1 Star

The star topology is characterized by having a central node that receives the information of the rest of the nodes, and all of them are directly connected to it. In networking terminology, the central node is usually called a hub. In this topology, all the information goes through the hub, even communications between the different clients in which the central node acts as a signal repeater. For the purpose of this thesis, the Android device that has to be connected to the network is linked to the central hub, which would have the information of all the nodes.

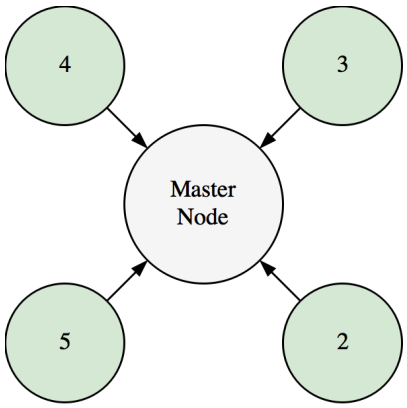


**Figure 3.1:** Star topology.

## 3.2 Ring

A ring network is the one in which all the nodes are connected in closed-loop configurations. The communications between nodes go through every intermediate element of the network until the destination is reached. All the nodes act as both server and client, and are able to repeat the signal until it reaches the destination. If two or more elements fail, the network could be split into different subnetworks with no communication between them.

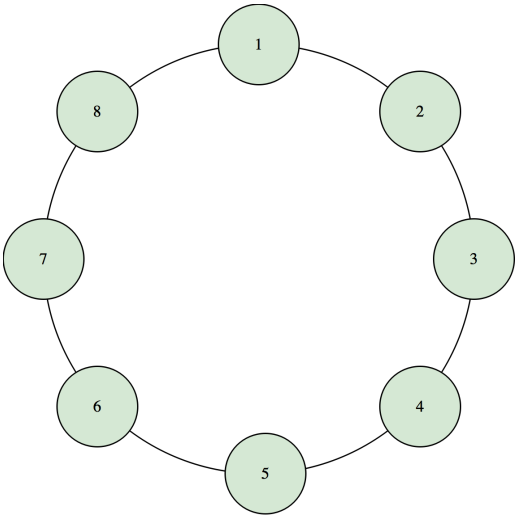


**Figure 3.2:** Network with ring topology.

## 3.3 Tree

The tree topology, also known as hierarchical topology, can be explained as a group of star networks arranged. Every individual node in the network acts as a receiver for the nodes connected to it and as a transmitter to the node to which it is connected. Theoretically, every single node could have as many childs as wished —the more added levels, the higher the number of nodes that can take part in the system.

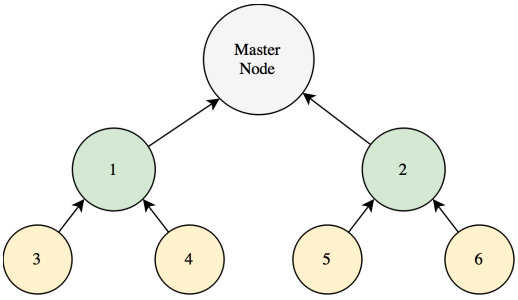


**Figure 3.3:** Network with tree topology, every node has two branches.

## 3.4 Mesh

In a mesh network, all the nodes cooperate to distribute data in the network, acting at the same time as servers and clients. Wireless mesh networks are usually called ad hoc network [19], because they do not rely on a pre-existing infrastructure and do not need to have a hub or central node. This type of networks are dynamic and nodes are free to move, what enables the creation and joining of networks anywhere and anytime. A mesh network can either be full mesh or partial mesh.

- **Partially-connected mesh:** Some of the nodes are linked together, and some of them can only be connected to only another one. This kind of network does not have the complexity and load to establish connections with every single node.

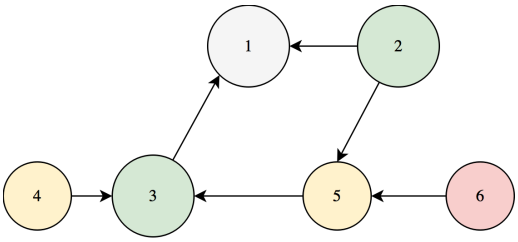


**Figure 3.4:** Partially-connected mesh network.

- **Fully-connected mesh:** All nodes are interconnected, so each node can send information directly to any element in the network with no intermediaries. This is highly not recommended when dealing with big networks.

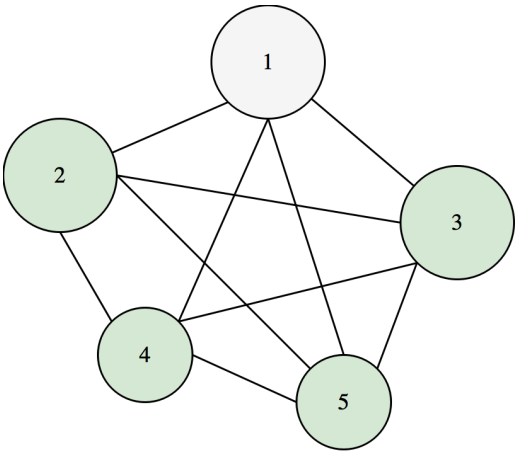


**Figure 3.5:** Fully-connected mesh network.

## 3.5 Connecting/Disconnecting

Connecting/Disconnecting refers to a common practice used in the Internet of Things applications, although it is not a network topology as such. The devices in a network can be independent and not connected to any other device until there

is a need of sending or receiving some information. Every device creates a network to which the rest of nodes can connect at any time, and disconnect whenever the information transfer has been done. Even if this method can be used for some specific applications, connecting one device to the network of another node takes some seconds and in some cases multiple tries until the connection is finally established.

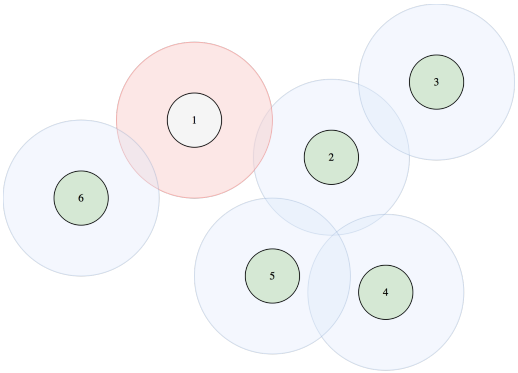


**Figure 3.6:** Connecting/Disconnecting type of network.

## 3.6 Analysis, comparison and assessment

All the topologies previously explained were at some point tested for the development of this thesis because of the requirements and needs to be met, as well as some changes and ideas suggested by the Supervisor and Co-Supervisor. Taking into account all of this, the final topology of choice was the mesh topology.

The main features that were considered in order to look for the best network topology for this project were the ones pointed out in Table 3.1. While doing the research, and as stated before, all of these topologies were tested to verify their viability.

| | **Ring** | **Star** | **Connecting Disconnecting** | **Tree** | **Mesh** |
|---|---|---|---|---|---|
| **Unlimited Nodes** | ✓ | | ✓ | ✓ | ✓ |
| **Decentralized** | ✓ | | ✓ | | ✓ |
| **Real Time** | ✓ | ✓ | | ✓ | ✓ |
| **Fast with high number of nodes** | | | | | ✓ |

**Table 3.1:** Comparison of network typologies.

A mesh network lets all the nodes to be connected to various devices, which makes communication faster and more robust, since upon losing the connection to the mesh by any module, any other of its sub-connections can be linked to the mesh. Due to the limitation of four devices per access point that the ESP8266 has, the star topology had to be discarded since more than 4 modules were used. Likewise, the main device has to handle all the connections as well as establishing the communication

with the smartphone, which can significantly increase the memory usage. The ability that this project has to auto-configure itself and being able to work in a decentralized way without relying in a pre-existing infrastructure is the main difference between the mesh and the tree topologies.

As explained later on, all the nodes in the system participate in routing by forwarding data to other nodes, so the structure is made dynamically based on the algorithm used. In this case, the modules connect to the access point with stronger signal with a maximum of four connections.

On the other hand, using a network with a ring shape can lead to communication problems and the delay of the messages, being these stuck up and sent every time it passes through an intermediary node. If there is an error in two points of the ring, the two independent networks will need to be configured again. Finally, the connecting/disconnecting mechanism was not chosen because of the time that it takes for one device to connect to a WiFi network. This delay of 1-2 seconds can make the system completely useless for applications in a real-time environment.

# 4
# Network Protocols

This section is intended not only to explain how all the protocols below work under the hood, but to expose some of their characteristics to the reader and the pros and cons that have led to the election of a particular one.

## 4.1 UDP

The User Datagram Protocol (UDP) is a low-latency internet transport layer protocol [2]. The main characteristic of the UDP is that it does not guarantee delivery and it is not protected against repetition of messages. When compared to other transport protocols, UDP is a connectionless protocol, which means that there is no end-to-end connection between the two devices trying to share information. This protocol does not verify the arrival of the message to its destination, what makes it lightweight by not ordering the messages nor tracking the receivers. This feature makes it ideal for internet video and audio streaming services [14]. Applications using this protocol need to deal directly with different things such as packetization, flow control and retransmission, what increases the development time.

- **Unreliable:** This protocol does not have handshaking dialogues, and thus the sender will not have any guarantee of the information being delivered.
- **Not Ordered:** If packages are delayed due to the internet connectivity or some blocking processes, the datagrams can be received in an order different from the transmission one. Applications need to handle these problems by themselves.
- **Lightweight:** The lack of reliability makes the packet sizes and overall load of this protocol smaller than others.
- **Broadcasts:** Data can be sent to all devices in a specific subnet.

## 4.2 TCP/IP

The Transmission Control Protocol (TCP) provides a reliable and ordered exchange of data between devices. TCP works with the Internet Protocol (IP), the latter dictating how the packets are sent out through the network, where they go and

how they reach their destinations. On the other hand, TCP ensures a reliable transmission of data across the connected networks, checking errors in the packets and retransmitting if needed [2]. Applications that need a notification every time a packet is sent, such as emails, the world wide web, and file transfers, are built on top of TCP [3].

Some of the main characteristics of this protocol are:

- **Reliability:** This protocol uses a number in order to identify each byte of data and the order of the bytes sent from each device so that the receiver can process the data in the correct order. Moreover, acknowledgements are sent by the receiver of the packet with a sequence number so that the sender knows that the information has been delivered.

- **Acknowledging:** The recipient of the message must send back a verification to the sender to let him know that the packet has been successfully received.

- **Data-Packaging:** TCP takes care of the data packaging and makes sure that data flows evenly and smoothly, dealing with problems that may arise halfway.

- **Multiple endpoints:** The connections established by TCP are identifiable, what allows each device to have various simultaneous connections to the same IP device or even to different devices, handling each connection separately without any conflict.

## 4.3 HTTP

HTTP stands for Hypertext Transfer Protocol, and the main difference with the other analyzed protocols is that it is an application protocol and therefore, of a higher level. HTTP is the main protocol for communication in the world wide web, used to exchange hypertext between points and built on top of TCP/IP (lower level). This protocol was the first one tested for this project since it is the easier and the most well-documented, but rapidly discarded for the problems that will be analyzed later. The ESP8266 devices include a support convenient for using HTTP, including libraries and already written code, which makes the development easier when trying to communicate between two different devices.

HTTP works as a request-response protocol with a client-server model, in which the client is connected to the server, delivering and requesting information. HTTP works by defining request methods (e.g. GET, HEAD, POST, PUT, DELETE) that can be used by the webserver to know what to do with the data received from the request. Although robust and reliable, this protocol uses hypertext data for communication, what makes packages large and communications slow. When working with embedded systems speed and small loads are a must, and HTTP can block the microprocessor for almost a second or more when sending and receiving

information. Furthermore, since it includes a web server and a client, the memory of the device is full so it can not be used for other needs.

## 4.4 MQTT

The Message Queue Telemetry Transport (MQTT) is a lightweight messaging protocol used in addition to the TCP/IP protocol. The main feature of MQTT is the low bandwidth required for sending information between nodes and the publish-subscribe messaging architecture used that requires an external broker. The broker is in charge of distributing the messages to the clients that are subscribed to a specific topic. Due to the growth of the Internet of Things, MQTT has become the protocol of choice for applications such as home automation or smart cities.

This communication protocol requires an external broker running in a computing device with operating system (e.g. PC or Raspberry Pi) that can handle the heaviest tasks, like distributing or ordering the messages, hence the clients do not need to know each other. Developing applications with MQTT can be done with very little effort thanks to the amount of libraries, documentation and big companies improving this technology.

## 4.5 Analysis, comparison and assessment

After testing and working with all the protocols mentioned before, the chosen one is TCP/IP. Below please find a table that summarizes some of the reasons which led to this decision.

| | **HTTP** | **MQTT** | **UDP** | **TCP/IP** |
|---|---|---|---|---|
| **Fast** | | ✓ | ✓ | ✓ |
| **Ordered Messages** | ✓ | ✓ | | ✓ |
| **External server needed** | | ✓ | | |
| **Reliable** | | ✓ | | ✓ |

**Table 4.1:** Main features analyzed for the chosen protocol.

In addition, average communication times were measured while testing the different protocols in order to have accurate speed data. This average is calculated after sending 100 messages from the smartphone device to the network using the specified protocols, and getting a response back from the device.

| | **HTTP** | **MQTT** | **UDP** | **TCP/IP** |
|---|---|---|---|---|
| **Time** | 1300 | 88 | 36 | 42 |

**Table 4.2:** Average time in milliseconds.

Lack of reliability is the main reason for not using UDP as a protocol of choice. As previously stated, when designing a gaming application where fast interaction is a

requirement, the sender of a certain package needs to know if the package has been delivered. For instance, if the user changes the game mode in the mobile application, the package including the information will be sent to the "master" node and instantly retransmitted to the rest of nodes. Before starting the game, the Android device has to be certain that the whole system knows which is the game mode.

If we compare TCP with a phone conversation, when talking to someone over the phone the caller can expect to always have an answer, and there is always feedback from both parts. As a metaphor, UDP could be compared to leaving a message in a voicemail, since there is no way to know if the receiver has received and listened to the voicemail or not.

On the other hand, MQTT needs to have a server as an intermediary so the mesh will not be fully independent. Likewise, it will need internet connection at all times to be able to share information between nodes. MQTT can be used for other type of applications in which the server can be used to make the complex calculations and organizing the heavy load in the network, while the devices have fewer and smaller functions. The time that it takes to send a message from the server to a single module is between 30 to 90 ms, and because of the concurrency of the servers multiples messages can be sent at the same time.

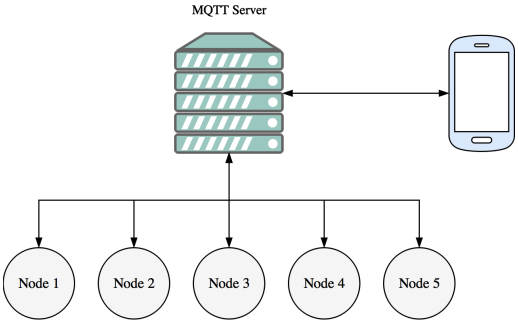


**Figure 4.1:** System layout with MQTT as a protocol.

HTTP is a protocol for which messages' sizes are big, and hard to work with because of its complexity. For this reason, and after testing HTTP as the protocol of choice for this project, the time for sending messages from one to another node was around 1-1.5 seconds. During this time, the main loop of the node was blocked and unable to make any other operation. Every single node acts as a web server and a client asking and sending data but the load of the data itself make the whole system unresponsive. HTTP and web servers can be very useful, but they were initially thought for handling websites and big chunks of data, which makes them not recommended for Internet-of-Things devices [10].

For all the reasons stated above, and trying to meet the requirements of a playware embedded system, TCP/IP is the protocol of choice. By looking at the tests, TCP is not the fastest protocol but it provides ordered and error-free data transfers and at the same time flow and congestion control.

# 5
# Hardware

## 5.1 The ESP8266

The ESP8266 is a chip with WiFi functionalities and a MCU (microcontroller unit) produced by Expressif Systems. As defined in the official website, a "low-power, highly-integrated WiFi solution". This chip is the most affordable one with all these capabilities and that is one of the reasons why it is widely used by board designers. It allows microcontrollers to connect to WiFi networks and make connections with Hayes-style commands [4]. In 2014, the company behind the chip released an official SDK that allows to program the chip in various languages, but the option used for this project was the Arduino Platform. Arduino is a C++ based open-source firmware that enables the ESP8266 chip and its microcontroller to be programmed as any other Arduino device on the market. During the development and research of this thesis, some other options appeared, such as the low-level SDK specific for mesh networks [6] by ExpressIf, but the lack of documentation and limited time made those options to be discarded.

| Category | Item | Parameters |
|---|---|---|
| **WiFi** | Protocols | 802.11 b/g/n/e/i |
| **WiFi** | Antenna | On-board, ceramic |
| **WiFi** | Security | WPA/WPA2 |
| **WiFi** | Modes | Station/SoftAP/SoftAP+Station |
| **Hardware** | Peripheral interface | UART/SDIO/SPI/I2C/I2S/GPIO/PWM |
| **Hardware** | Processor | Tensilica's L106 Diamond 32-bit |
| **Software** | Network Protocols | IPv4, TCP/UDP/HTTP/FTP |
| **Software** | Firmware upgrade | UART Download/OTA (via network) |

**Table 5.1:** Some relevant ESP8266 specifications

The ESP8266 is the core MCU of the two boards used for the development for the modular embedded system, the NodeMCU and the custom Tetra Boards.

### 5.1.1 Limitations

Although the ESP8266 is a powerful chip with almost all the WiFi functionalities needed, during the development of this project a limitation in the chips was found. This limitation changed completely the design of the network topology that was

thought of at first.

One ESP chip working as an Access Point (AP) can only handle 4 incoming connections. This limitation is a variable defined in the official ExpressIf SDK. The initial idea for this project was to use one of the nodes as a master working in AP mode, and connect the rest as a STATION (Station) to it. This would have led to a system with a maximum of five devices, limiting the future applications of this system. As explained in Chapter 3, the solution to this problem was to change the network topology to a mesh instead of a star, what made the system to not have a finite number of possible nodes.

Some other limitations appeared while working with these boards in terms of memory usage. It is highly important to keep the code efficient at all times and not to fill the memory with useless information, since communications between devices can be slowed down for this reason.

## 5.2 NodeMCU

NodeMCU is an open source IoT board based on the ESP8266 [20] chip and SDK from EspressIf Systems that helps to prototype IoT products. The firmware uses the Lua [20] scripting language, but thanks to the flexibility of the SDK it can run Arduino with ease. The NodeMCU is one of the most inexpensive boards on the market including all the features needed for fast development with the ESP8266 chip. Since it is an open source PCB, it is produced by many manufacturers. It also integrates GPIO, PWM, I2C, ADC and the FTDI or UART chip to handle communication between the computer and the board. As explained in previous chapters, the current state of open source electronic projects is making this kind of boards under constant development by the community, being high-quality documentation and example codes available for public use.

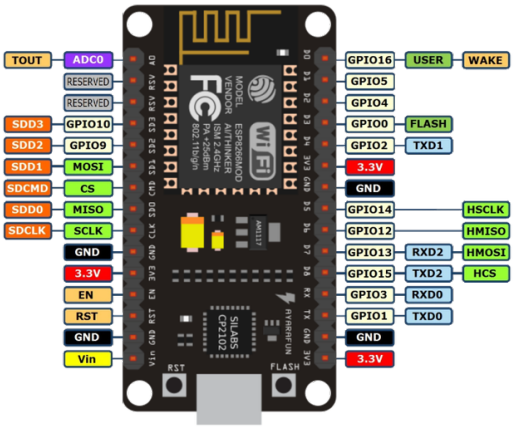


**Figure 5.1:** NodeMCU, board used during development.

The main reason to use this board during the development of the Thesis was the low cost of acquisition and the out-of-the-box features, since the programmer can start

programming and testing code as soon as they are plugged to the computer. The nodeMCU boards were mainly used to try WiFi functionalities and to test how the network should work. Then, the code base was moved to the circular tetra boards for the final demos. Furthermore, the final Tetra Boards needed a battery to be connected while flashing and programming them, and the flash button to be clicked every time it is done. Even if it might not seem an inconvenience, it was a considered as a limitation when working with a modular embedded system, testing more than 4 devices simultaneously and having to carry all the batteries.

## 5.3 Tetra Boards

The circular Tetra boards combine the capabilities of the ESP8266 chip with some actuators and sensors in order to be able to build a playful experience. Since these boards are custom made, there is no online documentation about them, but the schematics and pcb design was provided by the supervisors of this project. The schematics of these boards are included in Appendix B.

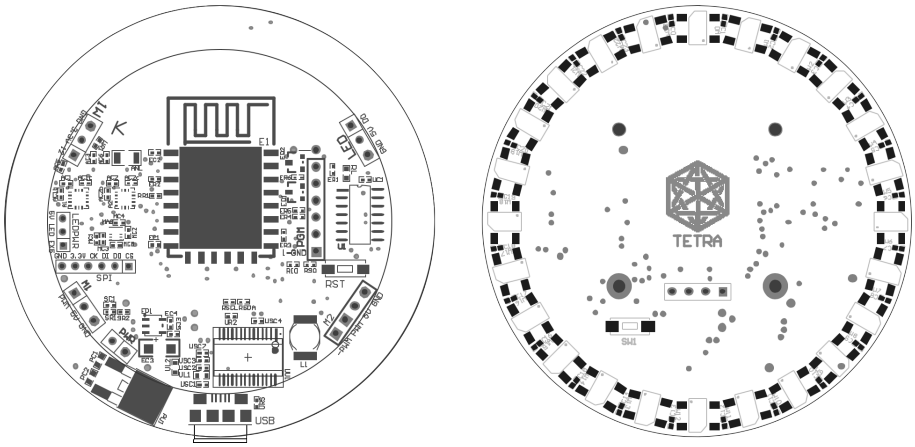


**Figure 5.2:** Tetra PCBs used for the project.

### 5.3.1 Features

These boards include all the elements of the ESP8266 chip as well as some other custom features:

- **MMA8451:** The board comes with two high-precision digital I2C accelerometers that can detect motion, tilt and basic orientation. These accelerometers are connected to the I2C pins of the ESP8266 (13, 14 PINS).

- **LEDs:** All the boards include the connections for a RGB led ring, but only some of them have all the leds mounted into the PCB. These leds can be used to create a huge variety of colors and effects using the NeoPixel Arduino library already written to help control all of them.

- **FTDI FT232RL:** It handles the communication between the micro-usb connector (PC) and the ESP8266 chip so it can be programmed and flashed. The FTDI company [7] also provides the drivers needed to program the chips and for the computer to be able to recognize the incoming connection.

# 6
# Software

This Chapter introduces the softwares used for the development of the project. We will explain how the Android Application and the program that runs in the boards work together to share information. All the code is available online (Please see Appendix A for the links).

## 6.1 Software used during development

As previously mentioned, the Arduino ecosystem was used for the development of this project with C and C++ as a language of choice. Arduino is the best framework to build electronics projects with ease, it includes lots of community and official pre-made libraries and takes care of compiling code into machine language. The hardware provided for the project is fully compatible with the Arduino firmware, and instead of using the official Arduino IDE for programming, Atom and its PlatformIO package was the first choice. Unlike the Arduino IDE, Atom provides the most common characteristics of any programming modern IDE, such as auto code completion, color highlights and a project files viewer.

Also during development, some other softwares were highly helpful for making this project possible, such as the official FTDI programmer. FT_PROG, as it is called by the FTDI company, is an utility useful for EEPROM programming FTDI devices, and it was used for some of the Tetra boards that came unprogrammed.

Android Studio, the official Android IDE, was also used for programming the Android Application explained later on in the next section.

## 6.2 Android Application

The idea behind creating an Android application is to test the functionalities and communications of the system to an external device, as well as being able to see some important data, share information and maybe to save the data in the phone storage in the future. The increasing capabilities of the smartphones produced nowadays make them suitable for multiple things, such as sharing the information of the modular network on the internet and thus releasing those heavyweight tasks from the embedded systems.

The possibility initially assessed to build the user interface for the system was AngularJS, a Javascript framework for building front-end applications. This option would have provided a multiplatform application suitable for phones, tablets and desktops at the same time, but the time constraints and a previous knowledge in Android programming made the author of this Thesis choose Android native development in JAVA as the most viable option.

The Android operating system (OS) provides all the development needed to make a fast prototype or even a final application to test the interaction of the users with the network and it is less restrictive than the main competitor iOS. Prior knowledge of Android programming and being able to access some of the low-level APIs for WiFi and TCP communications made it the perfect choice for this type of project.

The computing power that these small Android devices have nowadays make them an optimal option to handle the heavy workload, and the nodes in the mesh will only need to take care of the small calculations, the mesh communications and the sensor that each of them has.

### 6.2.1 How does the app work?

For the purpose of this thesis the Android device connects to the WiFi network created by the mesh. Since only one module can be connected to the external smartphone, that device is called the "Master" device. The rest of nodes in the mesh will know the unique ID of the Master device, and whenever they need to share information with the external device it will always go through the Master. The communication protocol between the smartphone and the mesh is TCP/IP, the one being used in the mesh itself. Once the Android device is connected to the network created by the mesh, it sends a message with its own information.

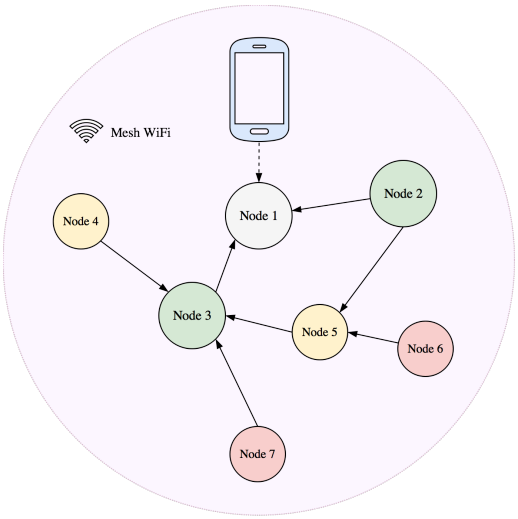


**Figure 6.1:** System layout.

The Android device is synchronized at all times with the mesh network and has all the information about it, including its structure. The master node sends information to it whenever a node is connected, disconnected or when the orientation of the any of the nodes has changed. All of the information that goes through the network to the Android external smartphone or tablet is formatted in the same type as the information is sent between the nodes, JSON.

The following line of JSON shows an example of a message sent by the application to the mesh, where it switches on the light of a specific node, in this case the ones with ID numbers 857489291 and 857480802. The `a` parameter stands for "activate" and it can be either 0 or 1.

```
[{
  "nodeId": 857489291,
  "a": 1
},{
  "nodeId": 857480802,
  "a": 1
}]
```

## 6.3 Embedded System Software

The purpose of this chapter is to explain in detail to the reader how the core code and the whole system work. Flowchart diagrams explaining the process and algorithms will be used in an effort to facilitate the understanding of the code. The approach followed for the design of the software that runs in every single node was to use a module structure in order to be consistent with the rest of the project's standards. Some of the questions that were raised during programming were:

- Which methods and functions should be placed into which modules?
- How does the data flow and which order should it follow?

Every module is located in a different folder in the file system, and all the classes and components needed for the module to work should be included in the same place. The code is planned keeping in mind the modules introduced in the next figure; some of them were deprecated in the final version although very helpful during development.

In order to be consistent with the Arduino ecosystem and standards, the main module classes (prefix `MC_` as a convention) always include two functions, `setup()` and `loop()`, the same ones that any Arduino sketch should include.

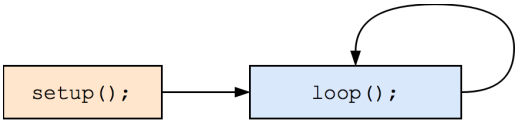


**Figure 6.2:** Basic Arduino functions.

- `setup()`: This function is called when the sketch is starting. It should set up everything that is later needed for the module to work and it is only called once.

- `loop()`: Once setup is called, the function "loop" is repeatedly executed in the main program until the program finishes or is restarted.

In the main.c class, the main setup and loop functions are called automatically by the Arduino firmware, and those functions should include the respective functions of rest of the modules. In every single module, other methods can be included below those two main functions to handle different functionalities.

### 6.3.1 Modular Cube

Modular Cubes (MC) is the name assigned to every single PCB with all its components included. The modular cube class has its own `setup()` and `loop()` that contains the respective methods of the different components, as explained later in Section 6.3.3. This class also includes all the general variables (including setters and getters) of the node so the rest of components can access and ask for any useful information. Some of those variables are:

- **tO:** Current time in ms since the moment the PCB was switched on.
- **deviceId:** Unique device ID that identifies the node in the network.
- **localIP:** IP of the node.
- **childs:** Contains a JSON formatted data of all the childs of the node.
- **master:** Boolean variable to know if the node is the master of the network or not.
- **currentOrientation:** Variable that gives a number between 1-6 indicating the current orientation of the cube. This variable is duplicated and it can also be accessed in the MC_Accelerometer module.
- **activated:** Boolean value that indicates if the device is on On or Off mode.
- **jsonData:** Information of the node and all its childs. This Json formatted data is slightly different depending on whether the device is Master or not.

### 6.3.2 Configuration

The configuration file <`Configuration.h`> is the one storing all the general variables for the system configuration, such as SSID or passwords, as well as some general server addresses. Having every single variable in the same place makes development and testing faster, since the programmer does not have to go through every file tweaking variables.

### 6.3.3 Modules / Components

As explained before, modules or components refer to the different parts of the boards themselves. Each of these components have an specific function in the software running inside the boards.

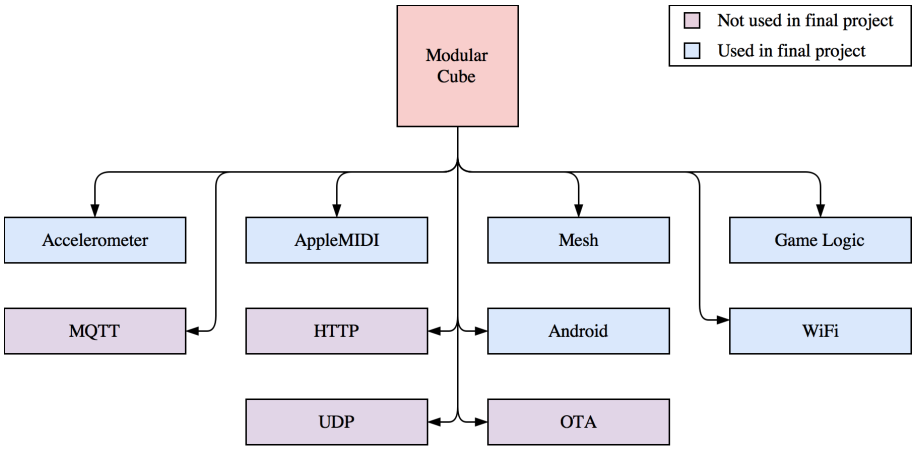


**Figure 6.3:** Components implemented inside each module.

#### 6.3.3.1 AppleMidi Support → [MC_MIDI]

The goal of this module is to be a wrapper for the official AppleMIDI Library [15], which enables Arduino to participate in an AppleMIDI Session, including all the Midi functionalities.

Midi (Musical Instrument Digital Interface) is a technical standard that describes a protocol, digital interface and connectors, and allows electronics musical instruments and other devices to connect and communicate with one another. MIDI handles messages that specify notation, pitch and velocity, and controls things such as audio values, panning, cues and clock signals. Using this library can make any Arduino device to be recognized as a digital instrument by a computer.

AppleMIDI (also known as rtpMidi [5]) is a protocol to transport MIDI messages within Real-Time Protocol packets over networks. In this specific case, wireless used by Apple. This protocol is free and open source and includes all the features that MIDI does.

In the setup function, an Apple Midi Session is created in the port defined by the user. Once this is done, the user can search for the device in the Audio Sound Manager in a Mac Computer or rtpMIDI software in Windows. Since the library includes a callback system, the user does not need to poll to receive the MIDI commands. On the other hand, in the loop function the AppleMidi library handles many other kinds of tasks, such as listening for incoming notes.

During development and testing a lag took place because of the PC being connected to the mesh network. The MIDI data being transferred through the system seemed to be too big for the modules to work properly, but when connecting the nodes to the home network where the PC or laptop was, the communications were real-time. In the Figure 6.4 it is shown how this system should work when being connected to a router with higher data transfer capabilities.

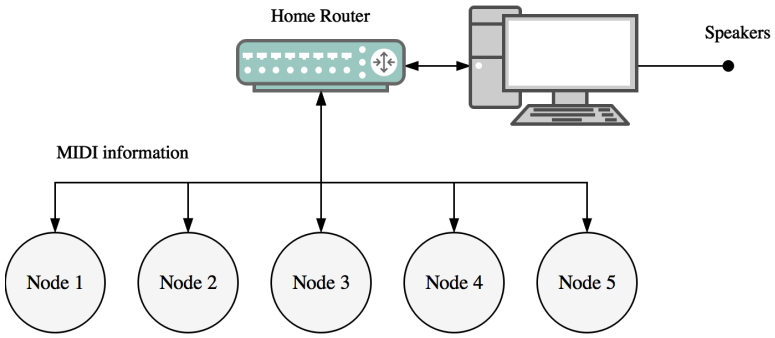


**Figure 6.4:** Viable MIDI system.

Eventhough AppleMIDI has not been used in the final application of the project, it would be interesting for a future project to expand in the use of MIDI and the connection of the mesh to a computer in order to use the cube modules as an instrument.

#### 6.3.3.2 MQTT → [MC_MQTT]

During the development of this project, MQTT was tested and finally discarded as it was explained in Chapter 4. This module uses and adapts the methods of the PubSubClient for handling MQTT messages between the broker and the master node that is connected to the internet. When the board is set up, the module handles the subscription to the specific topics, so that the broker knows where to send the information. In the loop function, it asks for new information to the broker and manages errors with the connection if they occur.

All the messages are in JSON format, so when data is received it has to be parsed and processed in order to determine which is the final receiver of the information. All the data received is then retransmitted to the final receiver specified.

Although this protocol has not been used in the final project, its speed, robustness and low-latency makes it suitable for other types of projects. For example, those in which it would be feasible to have a server as an intermediary between all the nodes in the network, including any Android device or computer.

#### 6.3.3.3 HTTP → [MC_Server]

Module in charge of the HTTP web server and client communication using the `<ESP8266WebServer.h>` library. A web server is created in the specified port and some data can be viewed if a phone is connected to the master just using the browser and visiting the default localhost IP 192.168.4.1. This module also handles the different types of HTTP request (GET, PUT, DELETE, POST) and parses the possible responses of those requests.

As mentioned in Chapter 4, HTTP was not used as a protocol for the current mesh configuration, thus this module is currently not being used in the final version.

#### 6.3.3.4 Over the Air updates → [MC_OTA]

Over The Air (OTA) updates is the process of loading the code into the board via WiFi without the use of any wires. This module handles this task and it is only used in development when all the boards are connected to the same home network. Both the Arduino and Atom IDE include this option, which is really helpful when trying to simultaneously update the code for multiple devices.

Since none of the devices were connected to the home network but to their own, it is not possible in the current mesh configuration to transfer over the air the amount of data needed for the code update. OTA updates would only be possible if the topology of the system is changed to one where the nodes are all connected to the same router.

#### 6.3.3.5 UDP → [MC_UDP]

UDP protocol was used during development and testing of this project and later discarded due to the justifications that was provided in Chapter 4. This module takes care of everything related with the UDP protocol, sending, receiving and parsing packets between nodes themselves and also between nodes and the Android device. UDP uses IP as a way to identify the nodes, usually the IP of the device, and an UDP server and client is set up in the defined port, in this case 5050. At first, the structure of the system required the devices to be linked by UDP, and the master node was connected to the Android smartphone vía an MQTT broker.

A low reliability and the need to code the way to handle all the errors was the main reason this module was finally discarded, although it was helpful for learning purposes.

#### 6.3.3.6 MMA8451 Accelerometer → [MC_Accelerometer]

The onboard MMA8451 accelerometer is set up and controlled by this module. During the setup, the sensor is detected and linked via the I2C protocol using the Wire.h Arduino library by knowing its address. Additionally, if no accelerometer is detected, the system automatically notifies the user in the command line, what is helpful if using the same code for the NodeMCU and the circular Tetra boards. This module makes use of standard I2C to communicate, read information and set variables on the sensor. All the information received regarding motion, tilt and acceleration are parsed and used for the control of the system, being the accelerometer the main input interface for the user.

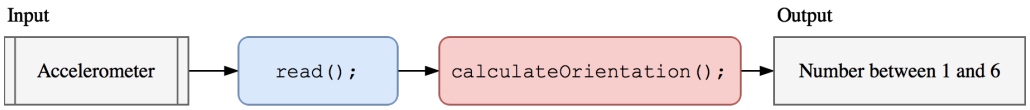


**Figure 6.5:** MMA8451 from the input to the output.

The MMA8451 sensor comes with some predefined functions that allow to know the

orientation of the node, although this function had to be adapted and calibrated for accuracy when using the cubes. In the code below we can see how the data is read and transformed into a valid current orientation for a cube, numbers between 1 and 6.

With the current demo, the data is read and parsed every 200 milliseconds. If the orientation changes in that time, the function `Cube.updateOrientation()` is run and the orientation is sent through the network to the Android device.

#### 6.3.3.7 WiFi → [MC_WiFi]

This is another wrapper that makes even easier to work with the default code in `<ESP8266WiFi.h>`. This module includes all the functions needed to set up the WiFi connection and to handle errors should they ever occur. Since its connection to the network is a core functionality of the system, the node will check on the loop if it is still connected to the internet. If not, it automatically reboots to find available networks.

This module was discarded since there is no external internet connection by the system anymore. All the network features are handled by the Mesh Module (MC_Mesh): connection, disconnection, routing, node communication, etc.

#### 6.3.3.8 Mesh Network → [MC_Mesh]

The Mesh module handles all the needs related to the connections and communications between devices in the network. The explanations in this section will aim to briefly cover how this part of the code works, and the thoughts and considerations behind this reasoning. This module has been the main focus during development because of the complexity of working with low-level communication protocols.

As explained in section 5.1.1, only 4 devices can be simultaneously connected to a node, so in order to connect other devices to the mesh we would need to configure a Mesh (as explained in chapter 3) that can be stable, robust and fast enough to meet the requirements of a real-time playware system.

As any other module in this system, the module has two main functions following the Arduino standards, `setup()` and `loop()`.

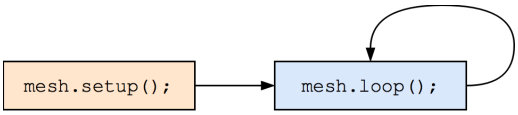


**Figure 6.6:** Mesh basic functions.

#### Mesh set-up

The first task needed when the device is switched on is to check if there is an available network around. The network is called “CUBES_MESH” by default, so

any device will look for that name and after gathering this information the mesh will be initialized.

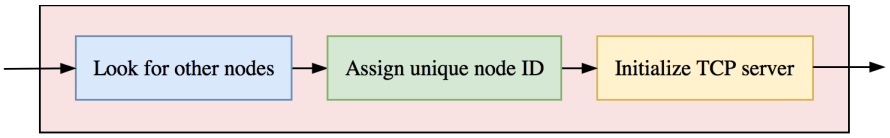


**Figure 6.7:** Mesh setup function.

Every new node is configured as both a Station (STA) and Access Point (AP), which lets this device connect via WiFi to other nodes and also be open in case any other node wants to connect to it. During the initialization process, an unique NodeId is given to each PCB based on the MAC Address. This ID will help the system to recognize any new devices, and also to be able to send messages to single devices.

After having used `encodeNodeId();` and giving it the MAC address of a device, it will return an integer that will be easier to handle by the system (Please see an example below in Figure 6.8).

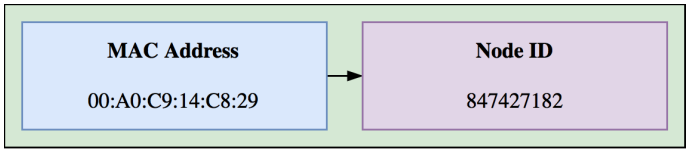


**Figure 6.8:** Encoding the MAC address to obtain a unique node ID.

In the process of setting up each device, we also initialize the TCP server that will handle the communications between the different elements in the network. The official Arduino `<espconn.h>` library has been used for this purpose. This library contains the general built-in functions that help us use TCP with ease, such as initializing the server or client, read messages and send messages from one TCP client to a server.

Although it may seem a very complicated process, all these functions explained above happen in just a matter of milliseconds. Once the setup has been done, we are ready to find any other nodes in the network, try to connect to them and start sharing information.

### Mesh loop

Every time the loop function runs, a decent amount of methods are called in order to make the system work. Two main functions are run over and over again inside the loop. Firstly, the WiFi and all the things related to the wireless connectivity need to be managed. Then we will manage the connections that are already taking place with any other device if any. Every node, even if connected to the network in the third or fourth level of the mesh, needs to know which is the NodeId of the master device that is connected to the smartphone or PC. For this reason, if the

current node does not know the ID of the master, it will ask for the master's ID to all of its connections on every loop and only if it is connected to the mesh.

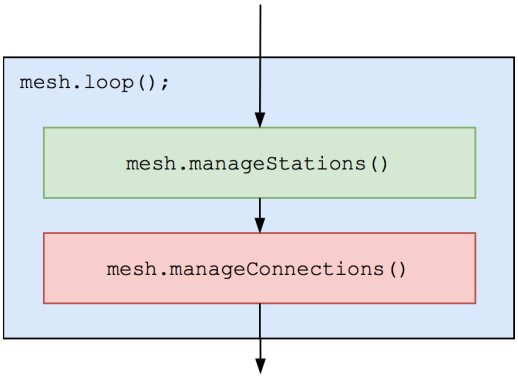


**Figure 6.9:** Inside mesh loop.

### Manage stations

The WiFi connections between devices are all taking place based on the callbacks that the Arduino WiFi library has. When there is any change in terms of the wireless connectivity, a callback is called and the WiFi status changes. This status can be easily accessed by using the `wifi_station_get_connect_status()` function and by parsing its response we can make changes to the device. We should bear in mind that we are inside the loop so this function will repeat itself, and by saving the previous status we can detect any change.

If the device is not connected to any Access Point (AP), it means that this specific node is not connected to the mesh network. Therefore, we need to start scanning for the mesh until we find another device nearby that we can connect to. By setting the network configuration as hidden we will get the data that it provides when scanning with the ESP8266 chips. Upon scanning for wireless networks, we will receive some useful information for every network such as:

- **SSID (Service Set Identifier):** Name of the wireless network. The name of the WiFi is not unique and can be repeated. As explained before, all the access points created by the devices in our mesh have the same name: "CUBES_MESH".
- **BSSID:** MAC Address of the access point founded, this is unique for every network founded.
- **RSSI (Received Signal Strength Indicator):** Is a measure of the power level of that radio-frequency device. The higher the level, the higher the signal strength to the specific network.
- **Security:** If the WiFi has any type of password we will the encryption type of it.

Using the table below as an example (Table 6.1), among the various options of *CUBE_MESH* available the one selected would be that one with RSSI of -21, since

is the one with the strongest signal. The current node will try to connect to that network next and if it fails, it will try again with the next one.

| SSID | BSSID | RSSI | Security |
|---|---|---|---|
| CUBE_MESH | 00:0a:95:9d:68:16 | -54 | WPA |
| HomeWiFi | 00:5a:10:1b:08:03 | -15 | WPA/WPA2 |
| CUBE_MESH | 00:23:60:aa:91:54 | -33 | WPA |
| CUBE_MESH | 00:11:93:99:77:21 | -21 | WPA |
| dtu | 00:1f:10:9c:45:03 | -83 | WPA2 |

**Table 6.1:** Example of a WiFi scan.

With all the information provided, each node is able to differentiate which of the networks nearby matches the name of the mesh, and out of those it can select which one has the best signal. The device will now try to connect to the mesh by using the device found as an entry point. If connecting results on failure, this network will be removed from the list of available networks to connect to and it will retry with the next one. As explained before, failing to connect to the network can occur because of the access point already having 4 nodes linked to it, thus exceeding the limit of connections.

Once the device is connected to the network, it will be assigned an IP. With that information and the Gateway IP address into which we are entering the network, TCP connection to the upper level node is ready to be started. As previously stated, the TCP setup takes place by using the low-level APIs that come available with Arduino. The callbacks that will help the device to know when a new message is received or whether a new node is connected to the mesh are now setup. The names given to the callbacks should be self explanatory of the moment when are they called:

- On successful connection
- On data received
- On data successfully sent
- On connection error
- On disconnection

Once the devices are paired, the device does not need to make any other changes regarding the WiFi. The client device sends the initial information about itself to the upper node it is connected to, which will now take it into account. Managing new connections and receiving data takes place in the next function in the loop, `manageConnections()`.

### Manage Connections

Each device calls `manageConnections()` in order to check and keep stable its connections to other nodes. Every single node saves all the connections to other devices into an array by using the library SimpleList [17], and on every loop it will iterate through this array of linked devices to check their status.

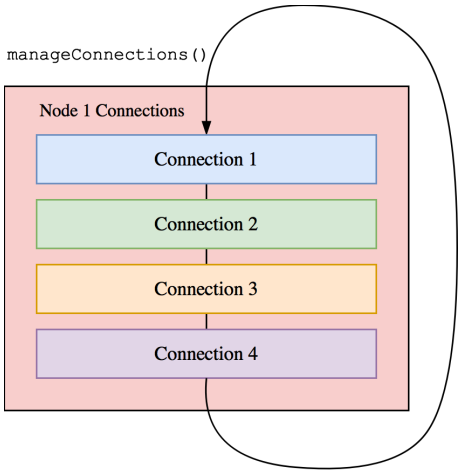


**Figure 6.10:** Components implemented inside each module.

All the nodes in the network are synchronized as a way of knowing which devices are currently on the mesh and which ones are not. That means that when cube module connects to the mesh it will receive information on what is the clock time of the mesh. All the nodes will have the same timestamp, which is useful to know when a device is disconnected or connected to the network and to verify that the connection is still alive. Figure 6.11 shows how the implementation works.

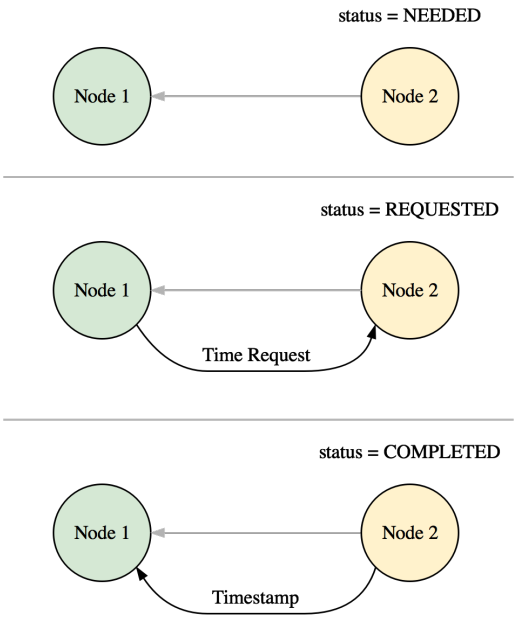


**Figure 6.11:** Node sync mechanism.

Every some seconds (determined in a variable), the device synchronizes the time of all of its current connections and sets their status to NEEDED. Then it sends sends a small packet to them and the connection status will now change to REQUESTED, and later on, once it gets a successful response from the device, it saves the time of the last received message and sets the status of that connection to COMPLETE.

If, by comparing the time in which we have received the last information from a node to the current time, the device finds out that this time is longer than the determined node maximum timeout, it will assume that this link is not alive anymore and will remove it from the array of connections.

```
    if (nowTime - connLastReceived > nodeTimeOut){
        // Connection dropped, remove node from the connection array
    }
```

Every node has a JSON formatted variable with the current structure of the network. This variable is updated every time there is a change in the network, so that all the modules know where they are located and which are the IDs of the rest.

```
[{
 "nodeId": 886361589,
 "subs": [{
     "nodeId": 886361061,
     "subs": [{
         "nodeId": 886360887,
         "subs": []
       },{
         "nodeId": 886360793,
         "subs": []
       },{
         "nodeId": 886360273,
         "subs": [{
             "nodeId": 886360526,
             "subs": []
           }]
       }]
   }]
},{
  "nodeId": 886360572,
  "subs": [{
      "nodeId": 886360555,
      "subs": []
    },{
      "nodeId": 886374821,
      "subs": []
    }]
}]
```

The code above shows an example of a mesh structure composed by 9 nodes. The *subs* include all the childs in that specific node.

### Sending data

Since all the devices in the network know their respective position in the mesh as well as the IDs of the rest of the devices, they can send messages to any device in the mesh. Sending data via the TCP protocol is straightforward by using the libraries included in Arduino. Here are two options available to send data to the network:

- Sending data to a single node: Each device needs to check if the destination node is in the lower levels, i.e., if it is any of the members of its connections or sub connections. If the receiver does not appear in this list, it will send the message to the upper level which will do the same process and check its own connections. This process will then repeat until the destination node is found and the message is successfully delivered to it.

- Sending data to all the nodes in the network: The node checks which devices are part of the network and sends an individual message to each of them.

Every message is sent using the JSON (JavaScript Object Notation) file format and parsed by the devices using the ArduinoJSON library [1]. All the messages contain the following information:

- Sender ID
- Receiver ID
- Timestamp
- Message

One example of a possible message shared between devices would be the following:

```
{
 "dest": 886361589,
 "from": 886360273,
 "timestamp": 17175046,
 "msg": {
    "t0": 17174790
  }
}
```

It is also important to mention that all the messages that are not sent to the receiver because of an error in the network are automatically stored in the queue. This queue will save the messages that need to be delivered and it will try to send them as soon as the receiver is available. In order to avoid filling the memory of the device with the queued messages, the queue will be cleared when the memory is full.

# 7

# Conclusions and future projects

## 7.1 Conclusions

In this Bachelor's Thesis, it has been analyzed, in summary, how a modular embedded system with the final goal of using WiFi technologies for playware applications should work. At a later stage, the code needed for the system to work has been developed in order to verify whether the research could be functional in a real-world implementation. All the aspects this Thesis involves have been tested in a real environment in order to choose the right solution for each specific problem.

After all of the analysis and development have been carried out, the main conclusions that can be drawn are the following:

- A wise and detailed choice on the network protocol to use can make a huge difference on how the system works. The lower-level ones, despite of being more complex to work with, are faster and more reliable. TCP/IP is not the fastest protocol but it provides ordered and error-free data transfers as well as flow and congestion control. Although some other network topologies could be used, the only one that meets the requirements of this system is the most complex to code, a partially-connected mesh.

- It is of particular relevance to get the most out of modular embedded system efficiency when designing the division of control between the modules and the smart device (i.e. Android smartphone).

- The ESP8266 chip favors the setup of networks of an almost unlimited number devices at a very convenient price. Having a decentralized network, where all the devices can act the same way, makes possible to connect various smart devices to the same network.

- Finally, WiFi technologies are a viable solution for real-time applications, and to be more specific, for playware ones. The Center of Playware can really benefit from the research carried out in this Thesis by using it in different future products and projects. As stated in the introduction of this document, the possibilities are endless, from modular robots connected via WiFi to handheld version of the MOTO tiles for exercising the upper part of the body.

## 7.2 Future projects

During the development of this research project, new ideas and study lines have arisen. These future projects could either be an extension of the current project or new fields of research to be promoted. The most outstanding are the following:

- Reduction of the size of the messages between nodes by changing the JSON format to bytes. The speed and robustness of the system will be increased and it will be really helpful towards the development of a final product using this technology.

- Implementation of all the functionalities that AppleMIDI offers (as explained in section 6.3.3.1) to make a full MIDI instrument based on the modular cubes as an interface.

- Research on how to use the Pulse Width Modulation (PWM) included in the Tetra boards to create moving modules with servomotors.

- Optimization of the code in order to be used in final user products by the Center of Playware, for example, a new version of the MOTO tiles working with WiFi.

# 8

# Bibliography

## References


[3] D.E. Comer. *Internetworking with TCP/IP:Principles, Protocols, and Architecture*. Fifth Edition. Pretince Hall, 2006.

[4] G. Dalakov. *The Modem of Dennis Hayes and Dale Heatherington*. 1999. (Visited on 05/29/2017).

[8] D. Groth and T. Skandier. *Network+ Study Guide*. Fourth Edition. Sybex, Inc, 2005.

[9] S. Inc. *Networking Complete*. Third. Sybex, 2002.

[11] C. Jessen, H.H. Lund, and T. Klitbo. *Playware - Intelligent technology for children's play TR-2005-1*. 2005. URL: `http://www.carsten-jessen.dk/playware-article1.pdf` (visited on 05/25/2017).

[12] J.D. Jessen and H.H. Lund. "Effects of short-term training of communitydwelling elderly with modular interactive tiles". In: *Games For Health Journal* 3(5) (2014), pp. 277–283. DOI: `10.1089/g4h.2014.0028`. URL: `https://goo.gl/3HNcAq`.

[13] J.D. Jessen and H.H. Lund. *Evaluation and understanding of Playware Technology – trials with playful balance training*. 2016. URL: `https://goo.gl/Jx64pi` (visited on 04/01/2017).

[14] J.F. Kurose and K.W. Ross. *Computer Networking: A Top-Down Approach*. 2010. URL: `https://goo.gl/Vb91us` (visited on 03/16/2017).

[16] Arnþór Magnússon et al. "Fable: Socially Interactive Modular Robot". In: *Proceedings of 18th International Symposium on Artificial Life and Robotics*. 2013.

[19] C.K. Toh. *Ad-Hoc Mobile Wireless Networks*. New Jersey, USA, 2002.


## Web resources


[1] B. Blanchon. *ArduinoJSON*. 2014. URL: `github.com/bblanchon/ArduinoJson` (visited on 04/20/2017).

[2] R. Branden. *RFC: Requirements for Internet Hosts - Communication Layers*. 1987. URL: `https://tools.ietf.org/html/rfc1122` (visited on 03/04/2017).

[5] T. Erichsen. *rtpMidi*. URL: `www.tobias-erichsen.de/software/rtpmidi.html` (visited on 02/25/2017).

[6] EspressIf. *ESP-MESH SDK*. 2016. URL: `espressif.com/en/products/software/esp-mesh` (visited on 03/16/2017).

[7] FTDI. *FTDI Corporate Information*. URL: www.ftdichip.com/FTCorporate.htm (visited on 03/04/2017).

[10] *IoT Standards and Protocols*. URL: www.postscapes.com/internet-of-things-protocols/ (visited on 05/29/2017).

[15] Lathoub. *AppleMIDI Library*. URL: github.com/lathoub/Arduino (visited on 02/23/2017).

[17] Phillaf. *SimpleList Arduino*. 2013. URL: github.com/Phillaf/Arduino-SimpleList (visited on 04/21/2017).

[18] Arduino Project. URL: www.arduino.cc (visited on 02/26/2017).

[20] Zeroday. *A lua based firmware for wifi-soc esp8266*. 2015. URL: github.com/nodemcu/nodemcu-firmware (visited on 03/02/2017).

# A

# Project's Code

All the code developed for this Thesis is published online divided in two sections: the embedded system and the Android application.

- **Embedded System:** `github.com/manugildev/modular-cubes-embedded`
- **Android Application:** `github.com/manugildev/modular-cubes-app`

# B

# Tetra PCB schematic