UNIVERSITY OF THESSALY

DIPLOMA

Indoor localization of smartphone devices by monitoring WiFi packets

Author:
Ioannis Goulias

Supervisor:
Athanasios Korakis
Associate Professor
Examiners:
Fotios Plessas
Associate Professor
Eleftherios Tsoukalas
Professor

A thesis submitted in fulfillment of the requirements for the degree of Diploma

in the

Network Implementation Testbed Laboratory
Department of Electrical and Computer Engineering

Volos, September 2019

"I have been impressed with the urgency of doing. Knowing is not enough; we must apply. Being willing is not enough; we must do."

Leonardo da Vinci

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

Περίληψη

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Διπλωματική Εργασία

Εντοπισμός **smartphone** συσκευών σε εσωτερικό χώρο με την παρακολούθηση **WiFi** πακέτων

Γούλιας Ιωάννης

Η διαδεδομένη χρήση των συσκευών που χρησιμοποιούν Wifi, έχει αποτελέσει σημείο ενδιαφέροντος για τον τομέα της έρευνας. Παρόλο που τα Παγκόσμια Συστήματα Θεσιθεσίας (GPS) είναι εξαιρετικά αποτελεσματικά σε εξωτερικούς χώρους, τέτοια συστήματα συναντούν δυσκολίες στην εφαρμογή τους σε εσωτερικούς χώρους. Για τον λόγο αυτό, τα συστήματα εντοπισμού εσωτερικού χώρου, χρειάζονται επιπλέον ανάλογες υποδομές. Σε αυτήν τη διπλωματική εργασία, θα αναπτύξουμε ένα Zigbee σύστημα εντοπισμού εσωτερικού χώρου. Αρχικά, θα υλοποιήσουμε έξι ESP sniffers που θα παρακολουθούν και θα καταλαμβάνουν τα Wifi πακέτα που μεταδίδονται στο εσωτερικό του εργαστηρίου. Στη συνέχεια, θα εγαταστήσουμε ένα Zigbee δίκτυο μέσω του οποίου τα ESP θα μεταδίδουν τη χρήσιμη πληροφορία από τα πακέτα που κατέλαβαν, στο gateway του συστήματος το οποίο είναι ένα beaglebone device. Σκοπός του beaglebone είναι να επεξεργάζεται τα δεδομένα που λαμβάνει και να ανεβάζει σε μια influx βάση δεδομένων τα σημαντικά στοιχεία όπως διευθύνσεις MAC, Received signal strength indication (RSSI) τιμές και το όνομα του κατόχου της συσκευής που εντοπίστηκε. Τελικώς, τα δεδομένα της βάσης δεδομένων θα μπορούν εύκολα από κάποιον υπολογιστή να επεξεργαστούν. Ελέγχοντας τις τιμές RSSI το σύστημα θα είναι ικανό να γνωρίζει την συσκευή ESP στην οποία βρίσκεται πιο κοντά η εντοπιζόμενη συσκευή. Το σύστημα θα αναπαριστά τις θέσεις των ενεργών και εντοπισμένων συσκευών μέσω ενός Web GUI.

UNIVERSITY OF THESSALY

Abstract

Department of Electrical and Computer Engineering

Diploma

Indoor localization of smartphone devices by monitoring WiFi packets

by Ioannis Goulias

The widespread use of WiFi-enabled devices has made indoor localization techniques a point of interest in the research field. Although global positioning systems are very efficient outdoors, such systems have difficulties in indoor localization applications. Therefore, indoor localization systems require additional indoor infrastructure. In this thesis, we deploy a ZigBee indoor localization system. Initially, we implement six ESP-based sniffers that capture the WiFi packets being transmitted by several devices in the interior of the laboratory. Afterwards we setup the ZigBee network through which the ESPs will transmit the useful information from the captured packets to a Beaglebone Device which acts as a gateway. The purpose of the Beaglebone device is to parse the data received and upload the important fields (such as the MAC address, the RSSI value and the name of each client) to an influx database. Ultimately, the data available in the database will be parsed by a computer deployed in the building. By checking the RSSI values, the computer will be able to point out the ESP device to which the device to be localized is closer. The system will demonstrate the localized clients with the use of a Web GUI.

Acknowledgements

Firstly I would like to thank my thesis advisor Prof. Athanasios Korakis for providing me with such an opportunity of getting involved into interesting projects that require special equipment not easily available to a student. This opportunity was really important for me.

Afterwards, I would like to thank the members of Nitlab, Polixronis Symeonidis, Nikos Sidiropoulos and Ioannis Kazdaridis for their valuable advice and hardware support.

Finally, I would like to express my gratitude to my parents and to my friends who truly cared for me and supported me constantly throughout my academic years. Without my parents my whole life would not exist, without my friends life would be tedious.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

The tremendous advance in the Internet of Things (IoT) applications and the extensive use of smartphones, has raised interest in indoor localization techniques. Since GPS signals are not efficient indoors[1], several approaches have been proposed to fulfill the task of indoor localization. Among these approaches, WiFi-based localization appears to be the most practical due to the fact that most buildings are equipped with WiFi access points[20]. Individually, WiFi localization implementations by exploiting the received signal strength indicator (RSSI) values appear to be much practical. In such systems, precision is the most crucial factor because indoor places are divided into various rooms and the localization has to be room-based in order to be efficient. The research in order to improve the accuracy of such systems will help individuals because of the many possible applications mentioned below.

Modern people tend to spend most of their time indoors. They also spend a significant amount of time using their smartphones and being connected to the internet through a Wifi network. Taking into consideration the facts mentioned above, indoor localization techniques could improve life quality. For example, in public places such as shopping malls, office buildings, and airports, precise indoor localization could be used for space management[19]. Moreover, in emergency events such techniques could help firefighters, police officers and medical staff to evacuate rooms and rescue people in congested buildings. In the medical field, indoor localization could provide medical staff with additional information about their patients. Through localization, they could monitor if a patient has not moved throughout the day, which is an indicator that the patient may not be feeling well.

1.2 System Architecture

In this thesis, we will implement a WiFi indoor localization system which utilizes the low-power 802.15.4 ZigBee wireless interface as a mediator[13]. Our implementation operates on ESP-32 system on a chip microcontrollers with modules of WiFi and Bluetooth. Those ESP-32 modules will be used as sniffers. Each time an ESP sniffer gets activated, it will transmit a packet to the gateway, requesting for the clients MAC addresses and MAC IDs registered in the building. The clients registered, are recorded in a text file located in the BeagleBone Black device which is used as a gateway. In this file the format is: (MAC AddressMAC IDClient Name). Hence, no commas or spaces are used. MAC IDs are matched to MAC addresses with ultimate

goal to reduce the overhead of the transmitted packets in the system. As soon as the ESP modules receive all information about the registered clients, they will be collecting packets from inside the building. They will be checking if any of those packets are transmitted by any of the registered clients. If that is the case, then from those packets, the ESP modules will gather the MAC addresses and the Received Signal Strength Indication (RSSI) of the clients based on their location. Accurately, the RSSI values will be computed by the ESP-32 modules and not by the access point that the client is connected to. Each minute, the ESP-32 modules will be sending the data gathered to a ZigBee end device through a serial port. Specifically, they will be transmitting packets with the format (MAC ID RSSI Value). In that way, instead of transmitting the whole MAC address which is six bytes long, only one byte is transmitted which is the MAC ID. The ZigBee end devices, configured to run on the same network, will be sending those data to the ZigBee coordinator. The ZigBee coordinator is connected to a BeagleBone Black device through a serial port. The data that the coordinator receives will be sent to the BeagleBone device. Afterwards, the BeagleBone device will connect to an Influx Database, set up on a laptop and will upload data corresponding to the data received by the ESP modules to the database. Such data are: MAC addresses, names and RSSI values of the active clients. Lastly, the laptop will be executing a script, that depending on the latest RSSI values posted on the database by the ESPs for each client, will point out the location of each client. Two sketches of the system architecture are shown in the following figures. The first one shows the procedure followed in order for an ESP module to obtain the MAC IDs and the MAC addresses of the registered clients. This procedure begins when an ESP module gets activated and ends when all client data are obtained. The second one shows the operational phase of the system during which, the registered clients get localized.

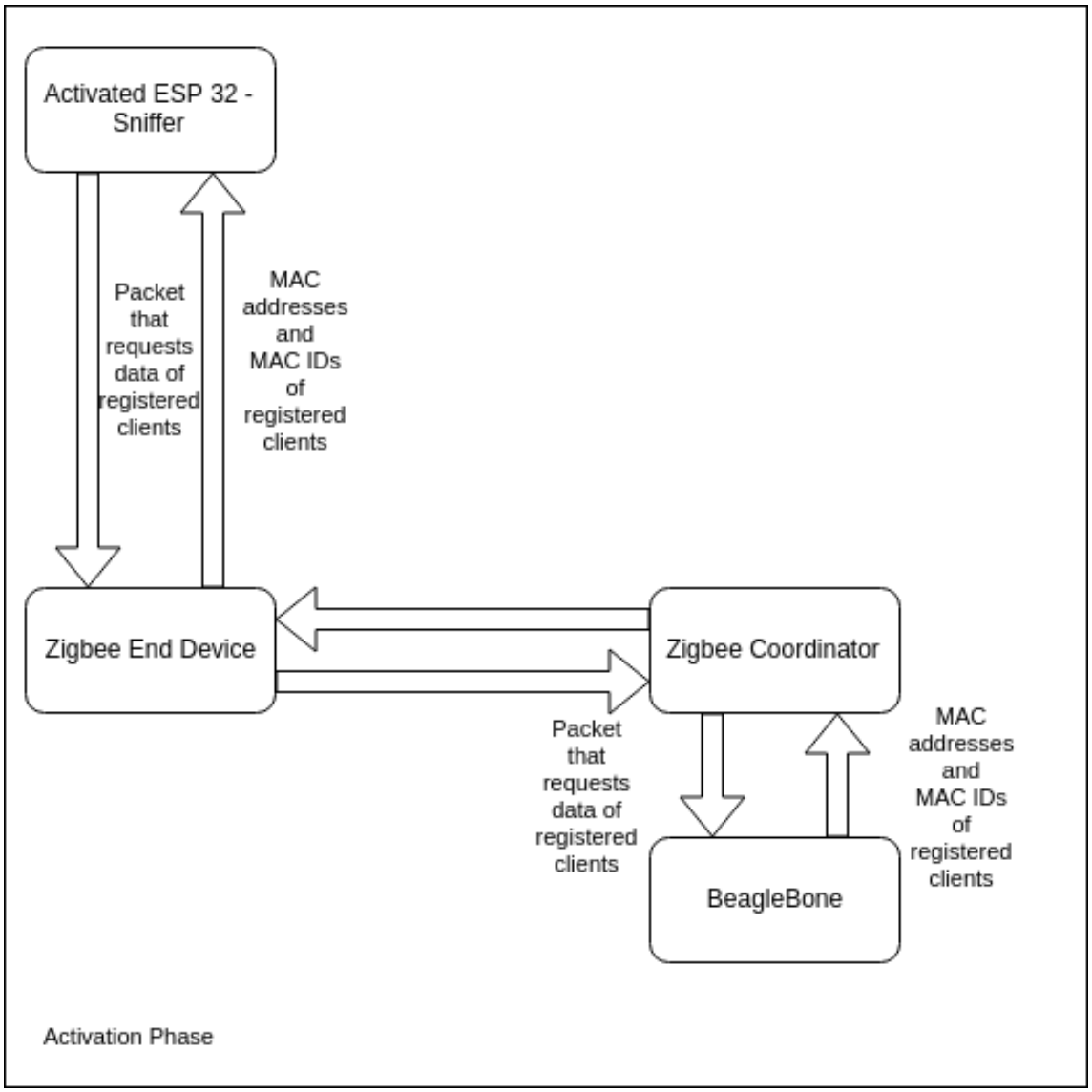

FIGURE 1.1: System Architecture Diagram.

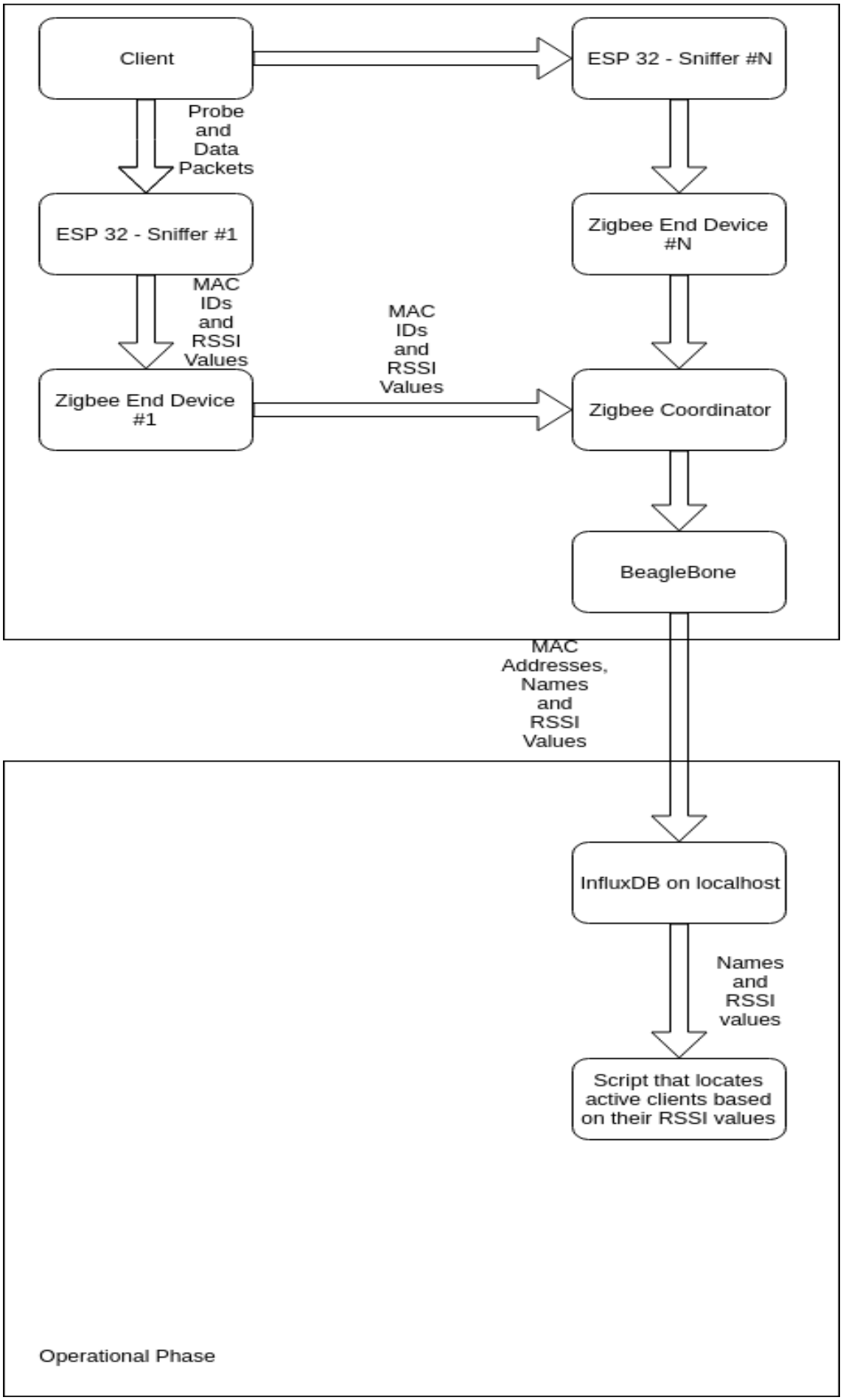

FIGURE 1.2: System Architecture Diagram.

1.3 Thesis Structure

Chapter 2 shows an in-depth analysis of the different WiFi frame types. It also points out which frames we found carrying valuable information for a successful client tracking implementation.

Chapter 3 introduces the ESP-32 Devkit V1 modules used in the system as packet sniffers. It also explains the meaning of a packet sniffer. Moreover, it demonstrates the code functions that are responsible for the localization of the clients.

Chapter 4 presents the Zigbee technology. Moreover, it demonstrates the XBee devices, their specifications and their usage in our system.

Chapter 5 provides information about the Beaglebone Black device. This device is used as a gateway in our system. Additionally, the chapter analyzes the functionalities and the program logic of the gateway in the system we implemented.

Chapter 6 aims to describe the way in which the data are extracted from the database, and edited in order to become the input of the web graphical user interface we developed. The chapter also, demonstrates the graphical user interface.

Chapter 7 describes the functionality of our implementation in its complete form.

Chapter 8 cites the results of the localization experiments we executed on the third floor of the laboratory.

Chapter 9 concludes with a summary of our implementation.

Chapter 2

WiFi 802.11 Frames

2.1 Overview

As already mentioned, our system will utilize ESP-32 modules as packet sniffers. These packets that the modules will be sniffing are 802.11 packets transmitted wirelessly between access points and clients. By capturing those packets, our system will be able to detect devices connected to an access point of the building. Afterwards, the system will be able to keep track of the devices, if those devices keep sending data to the network. The system will keep capturing the data and will be able to show the location changes of each device. Current IEEE 802.11 protocol, integrates data transmitted in a WiFi network into datagrams called frames. Datagrams are the basic transfer unit of data. They are structured in header and payload sections, whereas the header section contains additional information about the packet and the payload section contains the actual data to be transmitted. Frames are separated into three major types[9]: **Management Frames**, **Data Frames** and **Control Frames**. Each type will be analyzed subsequently along with the frame control field.

2.2 Frame Control Field

Frame control field is a header field present in all types of frames. It consists of two bytes and its details are shown in the following figure.

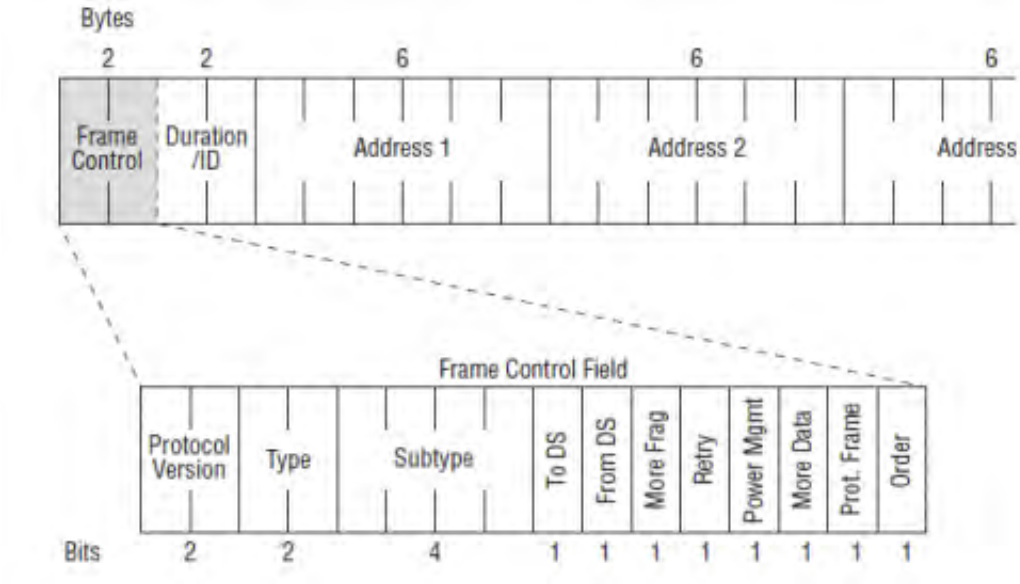

FIGURE 2.1: Frame Control Field[9].

Now we will decompose the frame control segments that we found useful in this thesis, in accordance with the figure above.

Protocol Version: This field is simply used to indicate which protocol version of 802.11 is used.
Type: Management frames use the 00 type identifier. Data frames use type field 10. Control frames use the type identifier 01.
Subtype: Due to the variety of frames, subtype field is used to differentiate even frames of the same type. Below is a figure that demonstrates different subtypes.

Type value b3 b2	Type description	Subtype value b7 b6 b5 b4	Subtype description
00	Management	1000	Beacon
00	Management	1010	Disassociation
01	Control	1010	Block ACK
01	Control	1011	RTS
01	Control	1101	ACK
10	Data	0000	Data
10	Data	0100	Null (no data)
10	Data	1000	QoS Data
10	Data	1100	QoS Null (no data)

FIGURE 2.2: Subtype Field[9].

2.3 Management Frames

Various types of management frames are used to provide services that on a wired network are simple. Features like identity establishment that in wired networks are easily implemented just by dragging wires are not so simple to implement in wireless networks. 802.11 protocol utilizes management features to handle such tasks. Services like searching for access points by clients, client authentication by the network and client to access point association are provided by management frames.

The structure of a management frame is shown in the following figure. The MAC header is the same in all management frames.

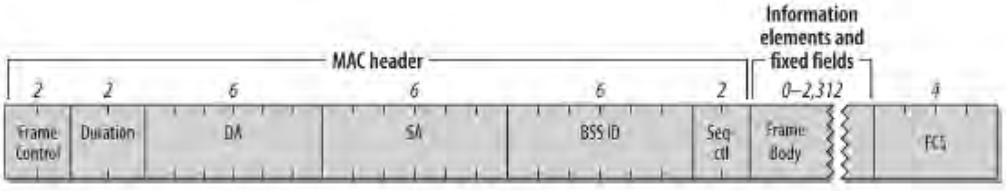

FIGURE 2.3: Management Frame[9].

2.3.1 Frame Control

In the system we catch and inspect only management packets with subtype 0100 (Probe Requests). The rest subtypes are not useful for client identification.

2.3.2 Addressing Fields

In this thesis, we found the addressing fields much useful in order to distinguish packets that clients are transmitting to the network, from other packets caught by the sniffers. Specifically:
Address 1: Address 1 is the receiver of the frame. The receiver is not always the destination of the frame. The destination is the station that will process the network-layer packet contained in the frame.
Address 2: Address 2 is the transmitter address. Transmitters are not always senders. The sender is the station originally generated the network-layer protocol packet in the frame.
Address 3: The Address 3 field is used for filtering by access points.

2.3.3 Frame Body

Management frames contain both fixed-length and variable-length fields in their body. From the numerous information provided by those fields, we found useful the Service Set Identity (SSID) field. SSID is used by the network managers to assign a letter-like identity to the basic service set (BSSID) of an access point. SSID is a string of bytes with maximum length 32 bytes, while BSSID is a 48 bit identifier of an access point. For an infrastructure BSS, BSSID is the MAC of the access point.

2.3.4 Subtypes of Management Frames

Beacon: Beacon frames show the existence of a network in the area. They are transmitted at regular intervals. Through their transmission access points can be identified by clients.
Probe Request: Mobile stations send probe request frames to scan an area for existing wireless networks. Clients keep sending those frames until they connect to a wireless network. Through those packets, our implementation can keep track of registered clients.
Probe Response: When a probe request reaches a network with compatible parameters, the network answers with a probe response frame. This frame enables clients to join the network.
Disassociation and Deauthentication: Disassociation frames are sent in order to finish an association relationship, while deauthentication frames are used for ending an authentication relationship.
Association Request: Through those frames the clients can join the network which enabled them to join with the probe response.
Association Response: When clients attempt to associate with an access point, they get an association response as a reply.

Summarily, our system takes advantage of management packets, specifically probe requests.

2.4 Data Frames

Data frames carry higher-level protocol data in their body. The following figure shows a generic data frame. In this section, we will inspect the fields of the data frames that we found useful for the implementation[9].

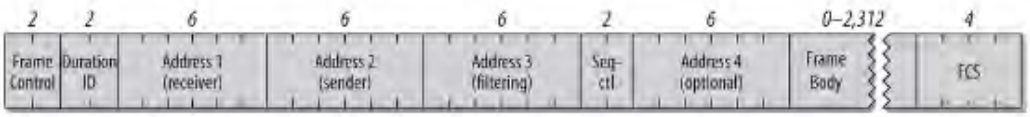

FIGURE 2.4: Generic Data Frame[9].

2.4.1 Frame Control

In the system we exploit data packets with subtypes 0000 (Data) and 1000 (QoS Data). The rest data subtypes (Null and QoS null) are used for power-saving status changes in the network and are not a valuable resource for tracking clients.

2.4.2 Addressing Fields

Addressing fields are identical to the ones of the management frames.

We utilized addressing fields of data packets to keep track of the clients in the building. As already mentioned, probe requests are only sent by the client device until the device connects to an available network. Through probe requests, we can only find the location of the registered client device when it initially connected to one of the networks. In order to keep track of the device, we constantly check the sniffed data packets. Precisely, while sniffing the packets, we search for data packets that their Address 2 field matches the MAC of a registered client. If such a packet is caught, then we know that this data packet was sent by one of the clients and we use that packet as an indication that the client is still present in the building. Then according to the calculated RSSI values of each sniffer, we update the location of the client. To conclude, data frames are the resource on which our system depends for keeping track of the registered clients.

2.5 Control Frames

Control frames assist in the delivery of data frames[9]. These packets handle the transmission medium. They all use the same control field, the one mentioned in the frame control field section. They are divided into the following subtypes:

Request to Send (RTS): Those control frames are used to gain control of the medium for the transmission of "large" frames. No data is transmitted in their body[9]. **Clear to Send (CTS)**: Those frames are used to answer request to send frames[9]. **Acknowledgment (ACK)**: ACK frames are used to send positive acknowledgments required for handshakes and successful transmissions[9]. **Power-Save Poll (PS-Poll)**: When a mobile station wakes from a power-saving mode, it transmits a PS-Poll

frame to the access point to retrieve any frames buffered while it was in power-saving mode[9].

It is essential to mention that we did not reclaim any control frames for client tracking in our implementation. None of the mentioned subtypes could assist us since those frames are not sent within a pattern but only on special events. They mostly assist the controlled communication between access points and clients in the network.

2.6 Summary

In this section, we saw the various types of Wifi frames. We analyzed the various types and subtypes. We speculated their content and we pointed out which frames introduced useful information about present clients in a Wifi network. We also stated the frames that our implementation exploits in order to provide localization of clients.

Chapter 3

ESP Devices - Sniffers

3.1 Overview

For the development of our system, six ESP32 development boards V1 were used. Those boards use the ESP32 WROOM 32[8]. Those boards form the nodes of the system. They keep watch of the WiFi packets that are exchanged through the network, they catch the useful data for the registered clients and they provide the measurements necessary for the localization of the users in the building. In the following sections, we will present the specifications of the boards used. Moreover, we will explain the programming logic we used on those boards, in order to be configured as sniffing-nodes. Finally, we will inspect some important details about the synchronization of those devices.

3.2 ESP32 Devkit V1 Specifications

The ESP32 Devkit V1 is a development board created by DOIT. It uses the ESP32 WROOM 32 microcontroller. It features WiFi and Bluetooth technologies. Below we demonstrate the specifications of the devkit[7]:

- Microcontroller: Dual-core CPU Xtensa LX6
- Operating Voltage: 3.3V
- Input Voltage: 7-12V
- UARTs: 3
- WiFi: IEEE 802.11 b/g/n/e/i:

An image of the devkit is show below:

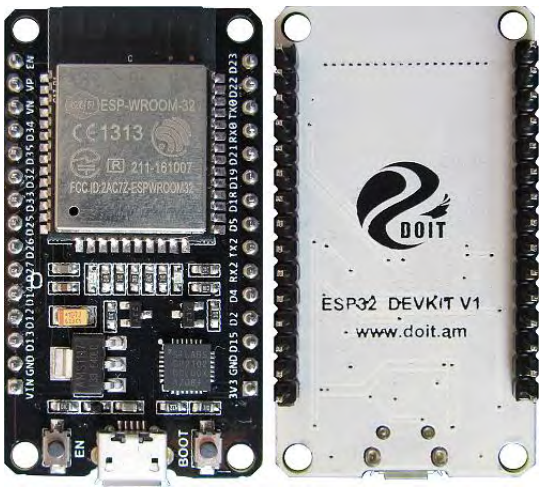

FIGURE 3.1: ESP32 Devkit V1.

For the power supply and the communication with the XBee end devices (the role of the XBee devices will be analyzed in chapter 4) we used the Vin, Gnd, Rx and Tx pins of the devkit. Below there is a picture demonstrating the pinout of the devkit:

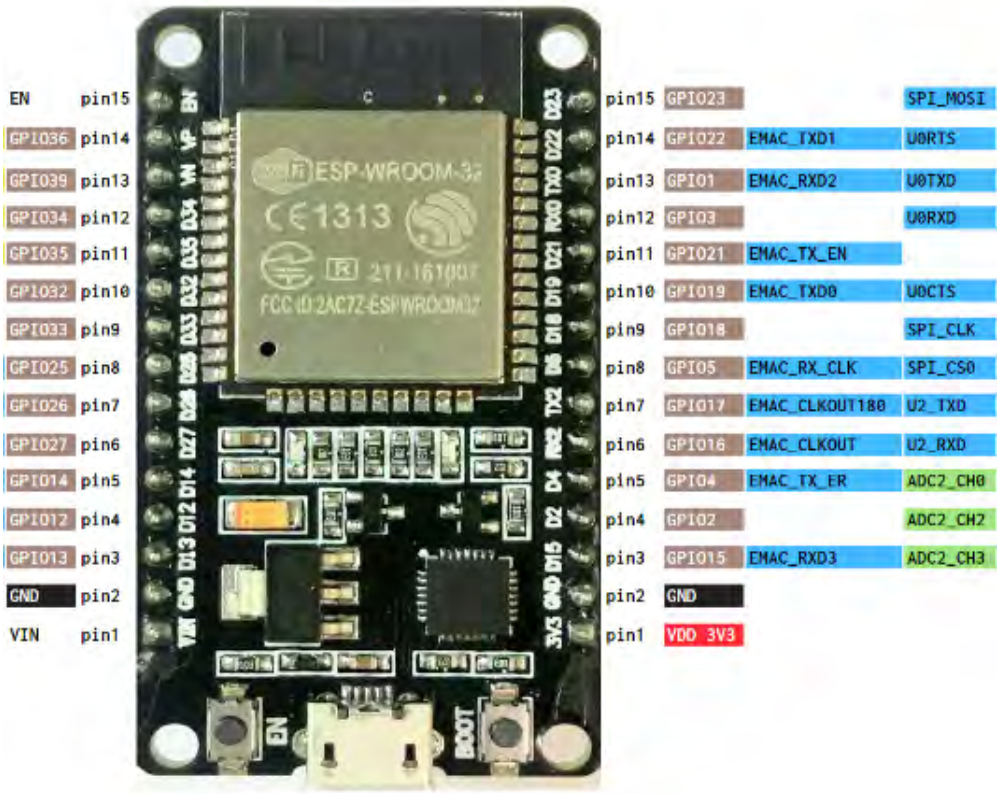

FIGURE 3.2: ESP32 Devkit Pinout.

Consequently, we used pins 1,2,12 and 13.
It is essential to state that the use of an ESP module is one of the most suitable options for the development of sniffer-nodes since their Wifi module can make use of

promiscuous mode. We will inspect this mode further on.

3.3 ESP32 Modules as Packet Sniffers

A packet sniffer can be a program or a hardware device that monitors the traffic of a network. A Wifi packet sniffer monitors the traffic of the Wifi networks in which the sniffer is deployed. In order to configure the ESP32 modules as Wifi sniffers, they have to be placed in promiscuous mode. Afterwards, they will be analyzing the packets exchanged in the network and extracting the useful information for the localization of the clients. In the following sections, we will demonstrate the programming logic and the functionalities of our packet sniffers.

3.3.1 Activation Phase

Clients present in the network can only be identified by their MAC addresses. The packet sniffers in our system, send data to the gateway regularly. Those data are encapsulated into frames. So in a simplistic case, we would be sending at least seven bytes of data for each active client. That is, six bytes for the MAC address and one byte for the RSSI value. In order to reduce that overhead, we introduced MAC IDs. Each clients' MAC address will be matched manually with a MAC ID. This is managed with the creation of a text file in the beaglebone device. This file, includes the MAC addresses, the MAC IDs and the names of the registered clients. With the use of MAC IDs, each time the packet sniffers will be sending only two bytes of information per active client. That is one byte for the MAC ID and one byte for the RSSI value.
Before entering the operational mode, the sniffers have to get information about which MAC addresses are registered for tracking and which MAC IDs are matched to those addresses. Thus, in the activation phase (the phase when a sniffer just got activated) the ESPs will firstly send a packet containing the data bytes 0xAA 0xCC. At this point, we should state that the communication between the XBee devices is carried out with the use of frames. So the data sent to the gateway by the ESPs are encapsulated into frames. Subsequently, the data received from the gateway are extracted from the frames received. This feature will be explained in chapter 4.
For the communication between the ESP and the XBee device, we used the library XBee.h. This library handles the connection of the serial port of the ESP. That is accomplished with the following commands:

```
XBee xbee = XBee();
xbee.setSerial(Serial);
```

The library is also responsible for creating the frames to be sent and extracting data from frames to be received.In order to create a transmit request, namely encapsulate the packet to be transmitted the library uses the command

```
ZBTxRequest zbTx_file=ZBTxRequest(addr64,payload_file,sizeof(payload_file));
```

It sends the packet with the command:

```
xbee.send(zbTx_file);
```

While, waiting for a received packet is accomplished with the following command:

```
xbee.readPacket(400);
```

The above command states that the serial will be waiting for 400 milliseconds in order to receive a packet.

Once the packet requesting the information of the file that contains the registered clients is sent, the ESP will be waiting for the packets that contain the requested information. The data field of the packets that the beaglebone device will send, have the following format: First byte is the byte 0xDD, second byte is the number of packets that consist the information of the file, rest of data bytes are the information about the clients. The ESP will be receiving those packets; it will check the second byte of those packets. If the number of packets received in the time interval of ten seconds, reaches the value of the second byte of the packets received, then all the information about the file will have been received. If more than ten seconds pass without the ESP receiving the announced number of packets, it will resend the frame requesting the file information. It will also set the counter of received packets to zero.
At this point, it is essential to mention that each ESP has a node ID. A value that represents the identity of each ESP. Starting from 0x01 and ending to 0x06. Depending on that value, initially the ESP will wait for a time interval of five seconds multiplied by the node ID. This feature is developed in order to avoid congestion in a case where all the ESPs get activated at the exact same time, for example, a comeback after a power outage.
While receiving the packets that form the file of the beaglebone, the ESPs will save that information to the following struct:

```
struct clients_known_array { //Struct that
  unsigned char clients_known[ETH_MAC_LEN];
  unsigned char time_passed;
  unsigned char client_rssi;
  unsigned char mac_hash;
};
```

FIGURE 3.3: ESP32 Clients Struct.

In the field clients known, the MAC address of each registered client is stored. The field time passed is used afterwards in order to point out which clients are active in the network. The client rssi field is used for storing the RSSI values of each client. Finally, the mac hash field is used for storing the MAC ID corresponding to the MAC address of the current client. Currently, the ESPs are developed to support up to one hundred clients. A received packet with information about the clients contains multiple clients, so the data are parsed accordingly by each ESP. Accurately, for each seven incoming bytes, the ESP will store the first six as the MAC address of the client and the seventh one as the MAC ID. This is demonstrated in the following segment of code:

```
for (int i = 2; i < rx.getDataLength(); i++) {


  if(k % 7 == 0) {
    clients_tracking_array[clients_known_count].time_passed = 0x00;
    clients_tracking_array[clients_known_count].mac_hash = rx.getData(i);
    //Serial.printf("MAC HASH %02x\n",rx.getData(i));
    clients_known_count++;
    k = 1;
  }
  else {
    clients_tracking_array[clients_known_count].clients_known[j] = rx.getData(i);

    j++;
    k++;
    if (j == 6){
      j = 0;
    }
  }


}
```

FIGURE 3.4: File Data Parser.

If all packets that include the file data are received, the ESP will proceed with the Wifi setup. It will initialize the Wifi module and then activate the promiscuous mode, which enables the ESP to monitor Wifi traffic. Another important feature is that before entering in promiscuous mode, each ESP will wait for a time interval of one minute minus the time that the node waited before transmitting the packet requesting for the file information. This feature was developed for synchronization reasons. We wanted all the ESPs of the system to get into promiscuous mode at the same time. After setting the Wifi module up, the ESPs will be ready for monitoring the network, gathering data and sending them to our gateway. That is the operational phase which is described in-depth in the following section. Below, are depicted the commands that enable the Wifi module and the promiscuous mode:

```
  /* setup wifi */
wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
esp_wifi_init(&cfg);
esp_wifi_set_storage(WIFI_STORAGE_RAM);
esp_wifi_set_mode(WIFI_MODE_NULL);
esp_wifi_start();
delay(60000 - (node_ID*5000));
esp_wifi_set_promiscuous(true);
esp_wifi_set_promiscuous_filter(&filt);
esp_wifi_set_promiscuous_rx_cb(&wifi_sniffer_packet_handler);
esp_wifi_set_channel(curChannel, WIFI_SECOND_CHAN_NONE);
```

FIGURE 3.5: ESP Wifi Setup Commands.

3.3.2 Operational Phase

As soon as the Wifi module and the promiscuous mode of an ESP get activated, the ESP begins capturing the transmitted packets. The command:

```
esp_wifi_set_promiscuous_rx_cb(&wifi_sniffer_packet_handler);
```

sets the function inside the parenthesis as a callback. That means that whenever a packet is captured this function is called.
Now, it is critical to refer to the libraries used on our code for the inspection of Wifi packets. Those libraries are natively developed for the ESP 32. We used those commands to include the libraries.

```
#include "esp_wifi.h"
#include "esp_wifi_types.h"
```

By enabling those clients, we were able to save the information of the captured packets in the following structs:

```
typedef struct { //struct that contains information of sniffed packets mac header
  unsigned frame_ctrl:16;
  unsigned duration_id:16;
  uint8_t addr1[6]; /* receiver address */
  uint8_t addr2[6]; /* sender address */
  uint8_t addr3[6]; /* filtering address */
  unsigned sequence_ctrl:16;
  uint8_t addr4[6];
} wifi_ieee80211_mac_hdr_t;

typedef struct { //struct that contains information of sniffed packets data
  wifi_ieee80211_mac_hdr_t hdr;
  uint8_t payload[0];
} wifi_ieee80211_packet_t;
```

FIGURE 3.6: Capture Packets Structs.

In the first struct, the header information is saved. The important fields are the source and the destination address of a packet. We can also extract the payload from the second struct. From that struct, we can inspect a packet and clarify its type. In our implementation, for the tracking of clients, we inspect only the probe requests and the data packets with source address the MAC address of a registered client. The ESPs capture all kinds of packets but the code inspects and parses only the packets just mentioned. We can recognize a probe request if a received packet is a management packet and if the first byte of its payload is the byte 0x40. If that is the case, then the ESP will look for the source address of the packet. Then, the ESP will search in the struct of the registered clients for that address. If a match is found, then that means that the probe request just captured is sent by a registered client. The ESP will update the time passed value to two. This means that the client will be considered as active for the next two minutes if no other data are received by that client for the next two minutes. In the case data from that client are received, the time passed field will be updated again to two. Moreover, the ESP will update the RSSI measurement of the client. So, each time we capture a probe request of a registered client, we get a new RSSI measurement which is essential for the regular localization. It is essential to state that we capture the absolute value of the RSSI and not the signed value of it. This is due to overhead reduction reasons. The code responsible for the features just described is depicted below:

```
else if(ppkt->payload[0] == 0x40) {//PROBE REQUEST
  //Serial.printf("PROBE REQUEST\n");
  int known = 0;   // Clear known flag
  for (int u = 0; u < clients_known_count; u++) //Check if client's MAC is in the clients_knowr
  {
    if (! memcmp(clients_tracking_array[u].clients_known, hdr->addr2, ETH_MAC_LEN)) { //if yes
      known = 1;
      clients_tracking_array[u].time_passed = 0x02; //time_passed gets value 0x02 so the client
      clients_tracking_array[u].client_rssi = abs(ppkt->rx_ctrl.rssi); //store the absolute val
      break;
    }
  }
}
```

FIGURE 3.7: Probe Request Handling.

In case of data packets, the exact same operation is executed. The only difference is that the ESP will check if the packet received is of type data instead of management. There is no need to check the payload of a data packet. The procedure is depicted below:

```
if (type == WIFI_PKT_DATA) {
  //nothing_new = 0;
  //Serial.printf("DATA\n");

    int known = 0;   // Clear known flag
    for (int u = 0; u < clients_known_count; u++) //Check if client's MAC is in the clie
    {
      if (! memcmp(clients_tracking_array[u].clients_known, hdr->addr2, ETH_MAC_LEN)) {
        known = 1;
        clients_tracking_array[u].time_passed = 0x02; //time_passed gets value 0x02 so t
        clients_tracking_array[u].client_rssi = abs(ppkt->rx_ctrl.rssi); //store the abs
        break;
      }
    }
```

FIGURE 3.8: Data Packets Handling.

Besides the promiscuous callback function, there are two more crucial functions to be inspected. The first one is named send clients. This function is executed every ten seconds. It is responsible for creating and sending the frames that contain the MAC IDs and the RSSI measurements of active clients. A client is considered as active if a packet with its MAC address as source was captured in the time interval of the past two minutes. The function firstly looks in the clients struct for the active clients. Afterwards, it prepares the packet data. First byte of data is the node ID of the ESP. The rest data have the following format: MAC ID RSSI. This format is used for each active client. A packet can contain up to forty-five clients. If more than forty-five clients are active then more packets will be sent. The code enabling that functionality is depicted below:

```
for(int p = 0; p < clients_known_count; p++) {
  if (clients_tracking_array[p].time_passed > 0x00 && clients_tracking_array[p].time_passed <= 0x02) { //if client is active
    present_clients_counter++; //counter that shows how many active clients are there currently
  }

}

number_of_regular_packets = present_clients_counter / 45; //each packet sent can contain up to 45 clients
//Serial.printf("number of regular packets %d\n", number_of_regular_packets);
last_packet_payload = present_clients_counter % 45; //last packet contains remaining clients

if (number_of_regular_packets > 0) {
  for (int q = 0; q < number_of_regular_packets; q++) {
    uint8_t payload[45 * 2 + 1]; //payload of a full packet containing 45 clients
    payload[0] = node_ID; //first byte of payload is the node_ID
    payload_counter = 1;

    while (payload_counter < 90) {

      if (clients_tracking_array[current_client].time_passed > 0x00 && clients_tracking_array[current_client].time_passed <= 0x02) {
        payload[payload_counter] = clients_tracking_array[current_client].mac_hash; //push MAC ID to the payload
        payload_counter++;
        payload[payload_counter] = clients_tracking_array[current_client].client_rssi; //push RSSI value to the payload
        payload_counter++;
        //Serial.printf("MAC HASH: %02x\n",clients_tracking_array[current_client].mac_hash);

      }
      current_client++;

    }
```

FIGURE 3.9: Send Clients Packet Preparation.

Once the payload of each packet is prepared, a transmit request will be created. Then, each packet will be sent to the XBee device connected to the ESP.

The second function of importance is called update clients known. This function is called every minute. For each active client, the function decreases the time passed value by one. So, if a client with the time passed value equal to two will not send any packets in the time interval of one minute, its time passed value will be decreased by one. If the same situation occurs in the next minute too, then the time passed value will be decreased again. Then the client will be considered as inactive. The crucial code of the function is depicted below:

```
for (u = 0; u < clients_known_count; u++) {
  if (clients_tracking_array[u].time_passed > 0x00 && clients_tracking_array[u].time_passed <= 0x02){
    clients_tracking_array[u].time_passed--;

  }
}
```

FIGURE 3.10: Update Clients Known Function.

3.4 Summary

In this chapter, we introduced the ESP-32 Devkit V1. We inspected its features and its pinout. We defined its role as a sniffer-node in our implementation. Finally, we explained and cited the code that those devkits execute in our system.

Chapter 4

XBEE Network

4.1 Overview

Our implementation is based on XBee devices for the transmission of captured packets by the ESP devices to the gateway. XBee devices use Zigbee technology. Firstly, we will explain what is the Zigbee technology and an XBee device. Afterwards, we will demonstrate how we configured the devices and what kind of information they transmit through their network in our implementation.

4.2 Zigbee Technology

Zigbee is a wireless technology developed as an open global standard. It is developed in order to be used in low-power IoT networks. Zigbee technology operates on the IEEE 802.15.4 physical radio specification, in unlicensed bands such as 2.4GHz, 900MHz and 868MHz. 802.15.4 specification is a packet-based protocol designed for low-cost networks. It allows devices to communicate through various topologies and have a longer battery life[6]. An important fact is that Zigbee protocol is able to support mesh networking. In such a network, nodes are interconnected and multiple pathways lead to each node. Compared to WiFi protocol, Zigbee has a significantly lower power consumption (25% of standard WiFi) and a lower data transfer (250kbit/s whereas WiFi has 54MBit/s)[4]. Zigbee systems could consist of three types of devices. Zigbee coordinator, router and end device. Each Zigbee network must have at least one coordinator. Coordinators are used for handling the data. Routers act as mediators that allow data to pass through them to other devices. End devices connect immediately to the devices that provide the data (measurements) to be transmitted. Such devices in our case are the ESPs. Our implementation uses only end devices and a coordinator.

4.3 XBee Devices

XBee devices, are a family of devices manufactured by Digi[5]. They can operate on a group of protocols. Zigbee is one of those protocols. Our system uses XBee end devices, each end device is connected serially with an ESP sniffer. It also uses an XBee coordinator that is connected serially to a Beaglebone black device. Each end device collects the data sent from the ESP sniffers. Then, all end devices send that data to the coordinator.

The XBee model that we used is the **XBee S2C**. It features[18]:

- Supply: 3.3V @ 33mA
- UART: 250kbps Max data rate
- Indoor Range: Up to 60m
- Outdoor Range: Up to 1200m

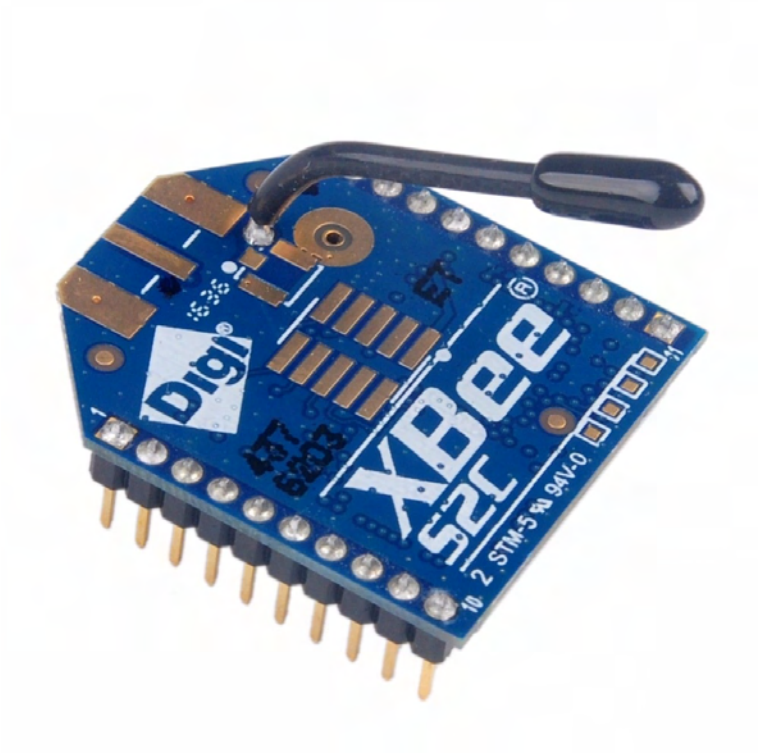

FIGURE 4.1: XBee S2C.

4.4 Configuring the XBees

In order to setup a mesh network composed of XBee devices, we used the XCTU program. A Digi configuration platform for XBee devices. This platform allows users to select through a variety of settings, in order to set up the XBee network. It can also be used for data monitoring or even sending data to XBees.

4.4.1 Operating Mode Configuration

An essential feature of the XBee configuration is the operating mode. This feature defines how data are transmitted between XBee devices. There are three operating modes:

Transparent (AT): In this mode, data are transmitted serially. Frames are not used; only pure data are transmitted in the network. Hence, there is no extra information such as headers during the transmission.

API 1: This mode uses a frame-based API. This means that the data to be transmitted are encapsulated into frames before their transmission. This mode is helpful for the safe transfer of data. Although there is more information to be transmitted because of the headers, this mode is more useful in cases where reliability and synchronization are significant.

API 2 Escaped Operating Mode: This mode is similar to API 1. The only difference is that API 1 relies only on the start delimiter and the bytes that show the frame length in order to differentiate between frames. If some bytes of a frame are lost, the length bytes will be showing false count. In that way, not only the current packet will be lost but also the next one. API 2, on the other hand, uses escaping character sequences. This feature can be useful in noisy environments since specific data values are escaped and not interfered with the actual data. In order to escape a byte, 0x7D byte value has to be used followed by the byte to be escaped and XOR'd with 0x20. This attribute provides additional reliability because if an unescaped 0x7E byte is found, the system considers it as the start of a new frame and the data previously received will be discarded. The escaped characters are converted again to their original sequence on the receiver.

4.4.2 XBEE API Frames

Both API operating modes use the following frame skeleton[5]:
Start Delimiter: Byte 0x7E constitutes the start delimiter of the frame. New frames received can be detected by reading that byte.
Length: This field consists of two bytes. The first is the most significant byte, the second one is the least significant. This field shows the total number of bytes present in the frame.
Frame Data: This field contains the byte that shows the kind of frame and the information bytes to be transmitted through the XBee network. XBees use several kinds of frames. The kinds that we used are mentioned below.
Checksum: Checksum is the last byte and contributes in transmission reliability. It is calculated by the hash sum of the previous frame bytes except of the fields start

delimiter and length. Frames with incorrect checksum will be ignored by the XBee device.

It is critical to mention that our system uses only the following types of frames: During each transmission, the sender will create a transmit request (data frame byte 0x10). The receiver of the transmit request, will send the data received through the serial port by creating a receive packet (data frame byte 0x90). The receiver will also send an acknowledgment packet to the sender. Afterwards, the sender will create a transmit status frame (data frame byte 0x8B). This frame will be transmitted through its serial port, demonstrating to the device that the XBee sender is connected to, whether the transmission was successful or not[5].

4.4.3 End Device Configuration

Now we will demonstrate the settings of the end devices.

The following figure shows the end module in the XCTU menu. We can notice that it uses 115200 baud rate, no parity bit, one stop bit and its operating mode is the API 2 mode.

FIGURE 4.2: End Device Module.

The firmware used on the end devices is the newest Zigbee TH Reg 4060.

FIGURE 4.3: End Device Firmware.

Below we demonstrate the settings with which we configured the XBee end devices. PAN ID used is DAAA. In order for the XBees to belong to the same network, they must all have the same PAN ID.

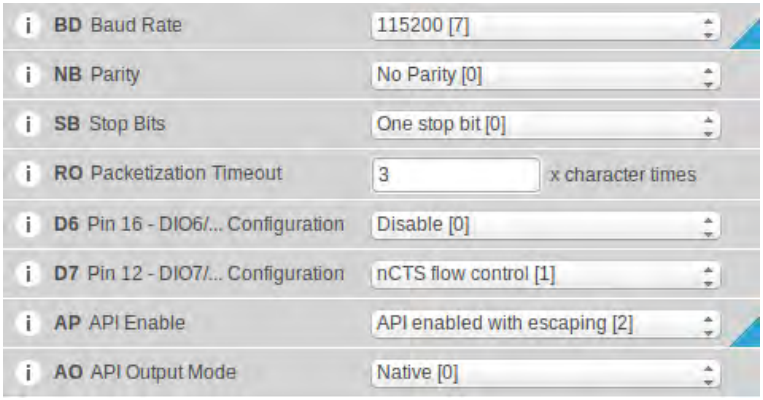

FIGURE 4.4: End Device Settings.

4.4.4 Coordinator Configuration

Subsequently, we will show the settings of the coordinator.

End devices and coordinator must have the same transmission settings, so again a baud rate of 115200 is selected. There is no parity bit, there is one stop bit and the mode is application transparent (AT).

FIGURE 4.5: Coordinator Module

The firmware used on the coordinator is the newest Zigbee TH Reg 4060.

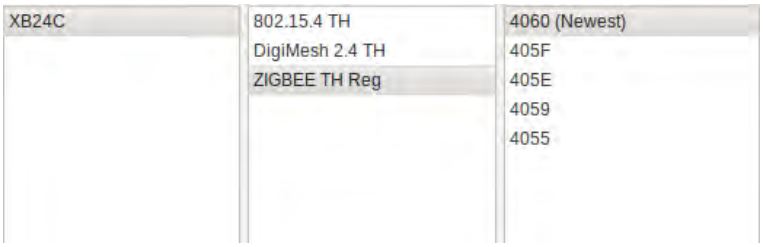

FIGURE 4.6: Coordinator Firmware.

For the establishment of communication, coordinator must also have the same PAN ID as the end devices. That is DAAA in our case.

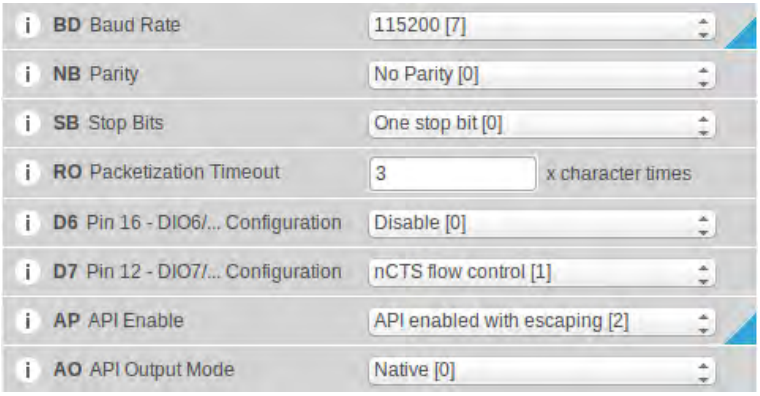

FIGURE 4.7: Coordinator Settings.

We shall point out that in order for a module to act as a coordinator, the selection below has to be activated:

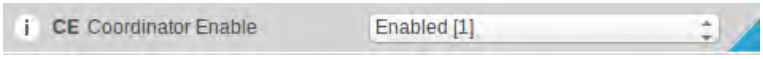

FIGURE 4.8: Coordinator Enable.

4.5 XBee Network in the Implementation

Our system uses seven XBee devices. One coordinator and six end devices. The coordinator is connected through UART port ttyO5 to a Beaglebone Black device. The role of the Beaglebone Black will be decomposed in the following chapter. Each of the six end devices is connected through a serial port to an ESP device. The functions that the XBee network serves are two. During the activation phase of the system, each end device will receive from an ESP a frame that requests the information of the file in which the clients are registered. This file is located in the Beaglebone Black device. Each end device will be responsible for transmitting that frame to the coordinator. Once this frame reaches the coordinator, it will be decomposed. The Beaglebone device will receive the frame data and the sender address and it will respond with the information of the file. The beaglebone device will prepare the frames with the information of the file and the coordinator will be responsible for transmitting those frames to the end device that sent the request for the file information.
During the operational phase, the end devices regularly receive frames from the ESPs that they are connected to. Those frames include the MAC IDs of the active clients and the RSSI value for each MAC ID. The end devices are responsible for transmitting that data to the coordinator. Coordinator XBee will disintegrate the received frames so that the Beaglebone will be able to process the data.

4.6 Summary

In this chapter, we showed what Zigbee technology is. We demonstrated the XBee devices we used and their capabilities. We also saw the mechanism through which

data are transmitted through XBee devices. Finally, we mentioned the role of the XBee network in our system.

Chapter 5

BeagleBone Black Device - Gateway

5.1 Overview

A gateway in an internet of things system, is the device that provides the connection between nodes that send the data and the database or cloud. The gateway receives data from nodes and preprocesses them in order to finally upload them to a database or a cloud. In our system, we used the beaglebone black device as a gateway. In the sections below, we will inspect the capabilities of the device. We will also demonstrate the functionalities of the gateway in our system and the technical details that enable the gateway with such functionalities.

5.2 Beaglebone Black Specifications

Beaglebone Black is a low-cost development board. It features[3]:

- Proccessor: AM335x 1GHz ARM Cortex-A8
- RAM: 512MB DDR3
- Storage: 4GB 8-bit eMMC on-board flash storage
- Graphics: 3D graphics accelerator
- USB client
- USB host
- Ethernet
- HDMI
- 2x 46 pin headers

Below is a picture of the Beaglebone Black:

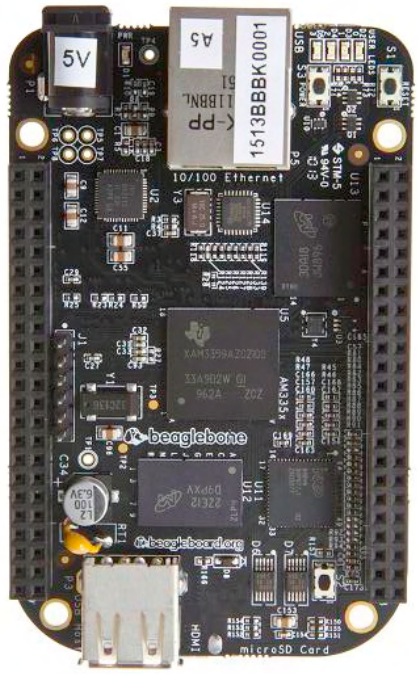

FIGURE 5.1: BeagleBone Black.

5.3 BeagleBone Black Setup

In this section we will mention the steps followed in order to flash an image to the beaglebone device and also make sure that the device will be able to access the internet every time we power it on. Additionally, we will configure the UART port responsible for communicating with the XBee coordinator device.

5.3.1 Flashing the Image

In order for the beaglebone device to be functional and programmable, we have to flash an image of a compatible operating system in it. In order to do that, we consulted the getting started guide of beagleboard.org[14] and browsed the latest firmware images of beagleboard.org[10]. We downloaded image Debian 9.5 2018-10-07 4GB eMMC IoT Flasher. Then, we loaded the image to an SD card. In order to do so we used the recommended program balena[2]. Afterwards, we inserted the SD card to the beaglebone device. We connected the beaglebone device to a laptop while pressing the boot button on the beaglebone device. After this step, the beaglebone started flashing the image. During this procedure the LEDs of the device were flashing sequentially. After 10 minutes the image had been flashed. We removed the SD card. We disconnected and then reconnected the beaglebone device to the laptop. The device was ready for use.

5.3.2 Establishing Internet Connection

In order to establish internet connection through USB for the beaglebone black, we had to route the device correctly through the network interface of the laptop we connected to it. We also managed to make those settings permanent by writing a new service for the operating system of the beaglebone. In order to access the beaglebone device through a terminal opened in our laptop, we used the command: "ssh debian@192.168.7.2". Debian is the default user created by the image we flashed. In laptop, we created a script with the following commands[15]:

```
#!bin/sh
iptables -A POSTROUTING -t nat -j MASQUERADE
echo 1 | sudo tee /proc/sys/net/ipv4/ip_forward > /dev/null
```

While on Beaglebone device, we wrote the following script named

`perm_internet.sh:`

```
#!/bin/sh
/sbin/route add default gw 192.168.7.1
echo "nameserver 8.8.8.8" >> /etc/resolv.conf
```

The execution of those two scripts enables the beaglebone black to use the laptop that is connected to as a router to access the internet.

5.3.3 Creating a Service for Internet Connection

Aiming to avoid the execution of the beaglebone script mentioned above every time we reboot the device, we created a service that executes that script on startup. Firstly, we moved the beaglebone script to folder /usr/bin of beaglebone. Then we executed:

`chmod u+x perm_internet.sh`

in order to make the script executable. We created the service by executing:

`nano /lib/systemd/perm_internet.service`

Pasting inside that service:

```
[Unit]
Description=Autostart Scripts
After=network.target

[Service]
Type=idle
ExecStart=/usr/bin/perm_internet.sh

[Install]
WantedBy=multi-user.target
```

Then we created the symlink and we enabled the service.

```
cd /etc/systemd/system/
ln /lib/systemd/perm_internet.service perm_internet.service
```

```
systemctl daemon-reload
systemctl start perm_internet.service
systemctl enable perm_internet.service
```

Beaglebone would now be able to access the internet through our laptop every time we booted on it, given the fact that we also executed the laptop's routing script. In order to reassure that, we executed ping google.com on beaglebone and saw that packets are exchanged.

5.3.4 Enable Beaglebone's UART Port

In the previous chapter, we stated that the XBee coordinator is connected to the beaglebone black gateway through a serial port. Beaglebone disposes UART serial ports. UART is a universal asynchronous receiver-transmitter, a computer hardware device for asynchronous serial communication[16]. By using UART, transmission speed can be configured. The beaglebone we used, uses a UART cape, it is depicted below. On that cape, the XBee coordinator can be placed. The cape uses UART 5 to communicate with the XBee. We enabled the UART 5 port by executing the following steps:

- Logging in to Beaglebone Black
- cd /boot
- sudo nano uEnv.txt
- Adding:

```
##Enable UART5
cape_enable=bone_capemgr.enable_partno=BB-UART5
##Overide capes with eeprom
uboot_overlay_addr3=/lib/firmware/BB-UART5-00A0.dtbo
```

- Saving the file
- Rebooting

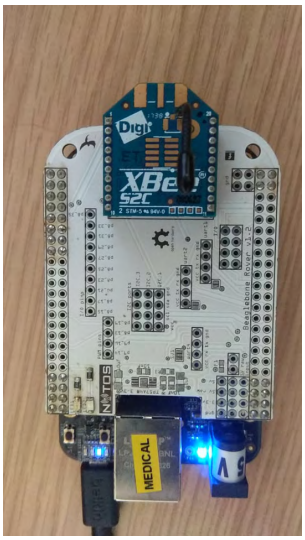

FIGURE 5.2: BeagleBone Cape.

5.4 Beaglebone Black as a Gateway

As we mentioned above, the beaglebone device is used as a gateway in our system. During the activation phase, beaglebone is responsible for transmitting a file with the registered clients to the ESPs. While during the operational phase, beaglebone acts as the gateway of the system, parsing data received from the ESPs and uploading them to a time-series database. Our gateway is programmed in python 3.5.3. In order to fulfill the functionalities of a gateway, beaglebone uses the digi.xbee.devices[17] and the influxdb[11] python modules. In the following paragraphs, we will inspect the functions mentioned above.

5.4.1 Activation Phase

In the beaglebone device, we have created a text file named macs.txt. Each line of the file represents a single client and has the following format: MAC MAC ID CLIENT NAME. No spaces are used in this format; also the MAC ID is the incrementing hexadecimal number for each line. For example, the first client has 00 as MAC ID, the second client has 01 as MAC ID and so forth. As we mentioned in chapter 3, MAC IDs were introduced to the implementation in order to reduce the overhead of the transmitted data. Without the use of MAC IDs, the frames that ESPs would be sending, would contain the whole MAC of each active client. That is six bytes. With the use of MAC IDs, only one byte per active client is transmitted from each ESP.
As soon as the python script located in the beaglebone device is executed, the device will read the file with the clients' information. For each line, it will append the MAC address and the MAC ID of the line to a string. Moreover, it will append the name to an array. Also, it will append the MAC to an array.
Once the file is read, the device will calculate how many packets will be sent once the ESP asks for the file information. Each packet can contain up to twelve clients, that is eighty-four bytes. Those packet use as a start delimiter the byte dd. As a second byte, they use the total number of packets that compose the file. In total, a full packet containing all twelve clients will consist of eighty-six data bytes. The data of each packet is saved to an array with the help of the string in which we previously appended MAC addresses and MAC IDs. Each row of that array represents the data of a packet to be transmitted with file information. Afterwards, the beaglebone will connect to the influx database that we set up in order to store the tracking data. It will also open the connection with the XBee coordinator through UART5 serial port. Finally, a callback function is called. This function is activated asynchronously, each time a frame reaches the coordinator. During the activation phase, the first frame to reach the coordinator by the ESPs will be the frame with data 0xAA 0xCC. As soon as this frame is received, the callback function will start sending the packets that correspond to the information of the file to the coordinator. The destination address of the frame will be the source address of the XBee that sent the 0xAA 0xCC frame.

5.4.2 Operational Phase

During this phase, the beaglebone device waits for data by the ESP sniffers. Each time that a frame with first data byte being 0x00 to 0x06 (the first data byte of those frames represents the sniffer node ID) is received, the callback function gets activated. It is important at this point to recall from chapter 3 that each data frame sent

by the ESP sniffers can contain up to forty-five clients. Each client uses two bytes of data. First one for the MAC ID and second one for the absolute RSSI value. Then, the data of the frame will be read character by character. For each client, which means every two incoming bytes, the script saves the MAC ID to a variable and the RSSI value to another variable. Due to the fact that for each client, the MAC ID is incremental, the script can easily access the corresponding MAC address in the array where all MAC addresses have been stored. In that case, the MAC of the first client registered in the file will be at the first row of the array and so forth. By accessing that array, the device will be able to retract the MAC address corresponding to the MAC ID received. After both the MAC gets retracted and the RSSI gets collected, the data for the current user will be posted to an influx database. We will analyze this kind of database in the following section.

5.4.3 Influx Database Use in Gateway

InfluxDB is an open source time-series database[12]. A time-series database stores time-series in pairs of timestamps and values. InfluxDB is created by InfluxData and it is used in various Internet of Things applications. There is interest in the way that data are posted to the database by the beaglebone. The part of the code responsible for the database posting is depicted below.

```
87      if ((data[0] == '0' and data[1] == '1')  or (data[0] == '0' and data[1] == '2') or (data[0] == '0' and data[1] == '3')):
88          in_frame_counter = 2
89          while(in_frame_counter < len(data)): #read data from packet
90              hash_mac = data[in_frame_counter] #first character of MAC ID
91              in_frame_counter += 1
92              hash_mac += data[in_frame_counter] #second character of MAC ID
93              in_frame_counter +=1
94              r = data[in_frame_counter] #first character of RSSI value
95              in_frame_counter += 1
96              r += data[in_frame_counter] #second character of RSSI value
97              in_frame_counter+=1
98              print("Host: "+data[0]+data[1] + " MAC: "+hash_mac)
99              hash_mac_number = int(hash_mac,16) #get int from hex form of MAD ID
100             mac_to_post = macs_array[hash_mac_number] #look up for the MAC address corresponding to MAC ID in the macs_array.
101             name_to_post = names_array[hash_mac_number] #look up for the name corresponding to MAC ID in the names_array
102             json_body = [ #json string containing the data to be posted
103                 {
104                     "measurement":measurements[int(data[1]) - 1],
105                     "tags": {
106                         "MAC":mac_to_post
107                     },
108                     "fields": {
109                         "Host":'1',
110                         "RSSI":r,
111                         "Name":name_to_post
112                     }
113                 }
114             ]
115             client.write_points(json_body) #write points to database
```

FIGURE 5.3: InfluxDB Posting.

As mentioned above, once the MAC address is found and the RSSI value is stored for each client of the incoming data frame, the beaglebone will post that data to the database. This is possible by creating a json body including the measurement of the database that the data belong to, the tag which is the MAC address of the client and the fields which are RSSI value and client name. In our implementation, the database

consists of six measurements. Measurements are the containers of the data. We used six since we use six sniffers. Each sniffer posts to a specific measurement.

5.5 Summary

In this chapter, we inspected the Beaglebone Black device, its capabilities and specifications. We defined the role of the gateway in our implementation. We also provided information about the configuration of the beaglebone device. Finally, we demonstrated the functionalities of the device in our system.

Chapter 6

Data Inspector - Web GUI

6.1 Overview

In the previous section, we explained that the gateway of the system posts the data gathered to an influx database. It is critical for every application that handles data to have a means of presenting those data. In our implementation, we used a parser for the influx database written in python. We also developed a web graphical user interface using html, javascript and python. We developed those tools in order to be able to allocate the clients into the rooms of the building. Each room has a sniffer-node. Hence, the node that reports the minimum RSSI absolute value for a specific registered client is closer to the client. That means that the registered client probably is located in the room that the sniffer-node belongs to. In the following sections, we will demonstrate the way that our parser handles the data. We will also demonstrate the web GUI.

6.2 Database Parser

The parser consists of two functions. Before executing those functions, it connects to the influx database we set up. The first function is named get users rssis. Every time this function is called, it will fetch data from the database for the time interval of twenty seconds before the call, until the exact time of the function call. The function fetches the data for all six measurements of the database, that is for all six sniffers. It fetches the name and the rssi of each client out of each measurement. If there are multiple entries for a client in a measurement for the last twenty seconds, only the latest one will be taken into consideration. The function stores the data into a python dictionary. The dictionary has key the name of the client and values the RSSI value from each measurement of the database. That is, at a typical case six RSSI values per fetched client name. In case that there is no RSSI entry from some measurements for a fetched name, then the value of the specified measurement will be assigned as "ff" which in terms of RSSI values is a maximum value and it is impossible to exist as a reasonable value in our system. The code of the function is depicted below:

```
def get_users_rssis(nodes=6):
    error_counter = 0
    rssi_per_name = {}
    for i in range(0,nodes):

        try:

            data = client.select_where('ESP', measurement_array[i], where='time > now() - 20s')
            #print(data)

            for item in data["results"][0]["series"][0]["values"]:
                #print(h)
                if (item[3] not in rssi_per_name.keys()):
                    rssi_per_name[item[3]] = ['ff']*nodes;

                rssi_per_name[item[3]][i] = item[4]
        except:
            print("There are no data available in the database for the specific time period.")
            error_counter += 1
            if (error_counter == nodes):
                return None

    return rssi_per_name
```

FIGURE 6.1: get users rssis Function.

The second function of the parser is named get users index. This function calls the get users rssis function and gets the returned dictionary. Then it accesses that dictionary and for each key, finds the minimum value of the RSSI values. We use the minimum value of the RSSI measurements in order to find the sniffer-node that the client is closer to. We can use the minimum value of the RSSI since we transmit the absolute value of the actual RSSI in the data. In another case, we would have to find the maximum value since the RSSI values would all be negatively signed. Once the minimum value is found, the function calculates the index of that value in the list of the six values per key. This means that for a client in the dictionary, the function will calculate the minimum RSSI value from the values of the specific client, if that value for example is the second in the list then the index result will be one (counting the index from zero). This happens for all the clients who are stored in the dictionary. Afterwards, the function will create a new dictionary named rooms dict representing the six rooms of the sector. As we mentioned before, in each room we have placed a sniffer-node. Afterwards, the function will assign the client names with common index to the according key-room of the dictionary. That means: If two clients have RSSI index zero, then they are probably located in the first room, if they have RSSI index one, then they are located in the second room and so forth. This will happen for all rooms. The names of each key-room are the values of the new dictionary. They will be sorted alphabetically and the function will return the dictionary. We cited the code of the function below:

```
def get_users_index(nodes=6):
    if (get_users_rssis(nodes) == None):
        return None
    names_rssi_dict = get_users_rssis(nodes)

    users_index = {}
    print(names_rssi_dict)
    for k in names_rssi_dict:
        rssi_list = names_rssi_dict[k]
        users_index[k] = rssi_list.index(min(rssi_list))

    rooms_dict = {}
    sorted_rooms_dict = {}

    for k in users_index:
        room = rooms_dict.get(users_index[k])
        if (room == None):
            rooms_dict[users_index[k]] = []
        rooms_dict[users_index[k]].append(k)
    for k in rooms_dict:
        sorted_rooms_dict[k] = sorted(rooms_dict[k])
    return sorted_rooms_dict
```

FIGURE 6.2: get users index Function.

It is essential to state that the data parser we just reviewed, is used by the web GUI. The web GUI regularly calls the get users index function. In the next section, we will demonstrate the web GUI and its development.

6.3 Web GUI

The web graphical user interface we developed, uses python flask, javascript and html. It constitutes a very practical way of demonstrating the live localization our system can provide. It is depicted in the following image:

FIGURE 6.3: Web GUI.

In order to develop it, we used two routes for GET methods in flask. The first GET method renders the bootstrap template we used. The second route is responsible for calling the functions of the parser. The second route returns a json object that

contains the rooms dict dictionary we mentioned before. Afterwards, the javascript that the GUI executes, will be calling the route that returns the json object every three seconds. With the data extracted from the json object, the javascript code will add content to the html body of the GUI. This content includes a column with the room name and the catalog of the clients located in that room. This content is added for each of the six rooms. We can see that also on the picture of the GUI cited above.

6.4 Summary

In this chapter, we inspected the database parser and its functions. We also clarified that the localization is based on the minimum measurement of the RSSI values for each client. Then, we demonstrated the GUI and the way it is updating its content.

Chapter 7

Spherical System Architecture Description

7.1 Overview

In the previous chapters, we described in depth, each component of our system. We demonstrated the role of each device and reviewed the code they execute. In this section, we are aiming to put those pieces together by describing the system as a whole.

7.2 The Devices

In this section, we will cite pictures of the devices used in the implementation: Below, we depict the ESP-32 Devkit V1 connected to an XBee end device.

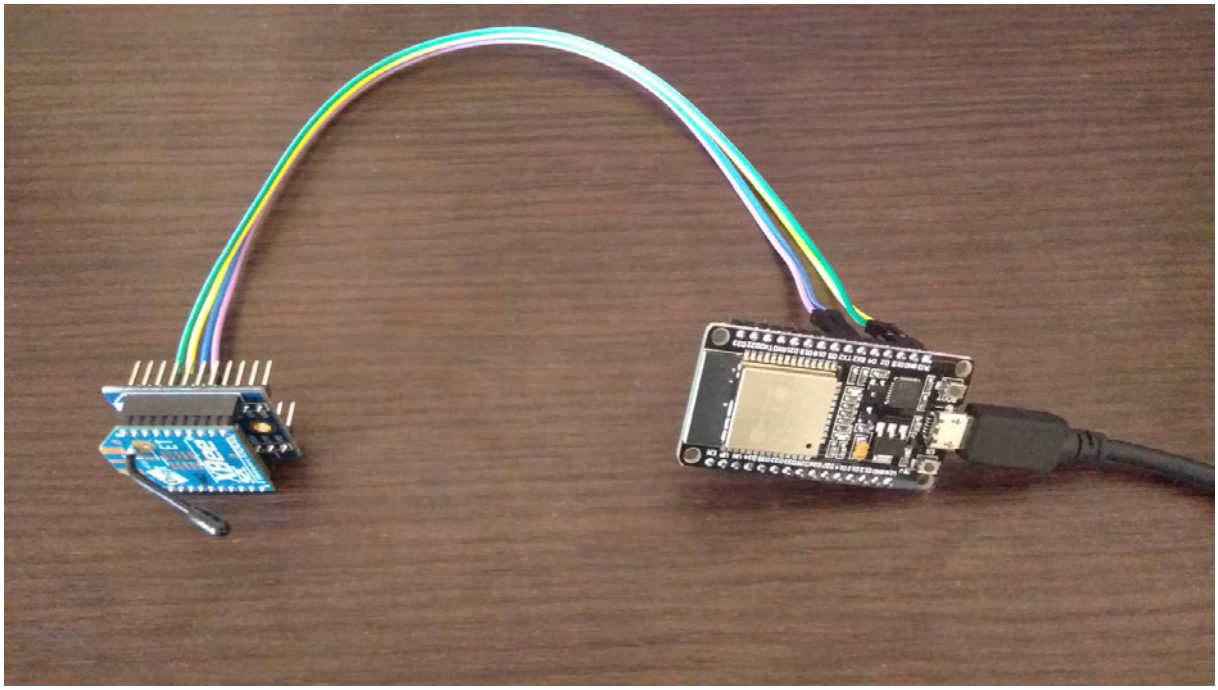

FIGURE 7.1: Sniffer Node.

This configuration, forms a sniffer node of the system. The ESP-32 is powered by a USB cable. The ESP-32 powers the XBee by two cables, one for 3.3V output and another for ground. The other two cables are responsible for the communication between the XBee and the ESP. The one is connected to the transmitting pin of the ESP and to the receiving pin of the XBee. The other one is connected to the receiving pin of the ESP and to the transmitting pin of the XBee. Our system disposes six sniffer nodes.

Below, there is a depiction of the beaglebone black device used as a gateway:

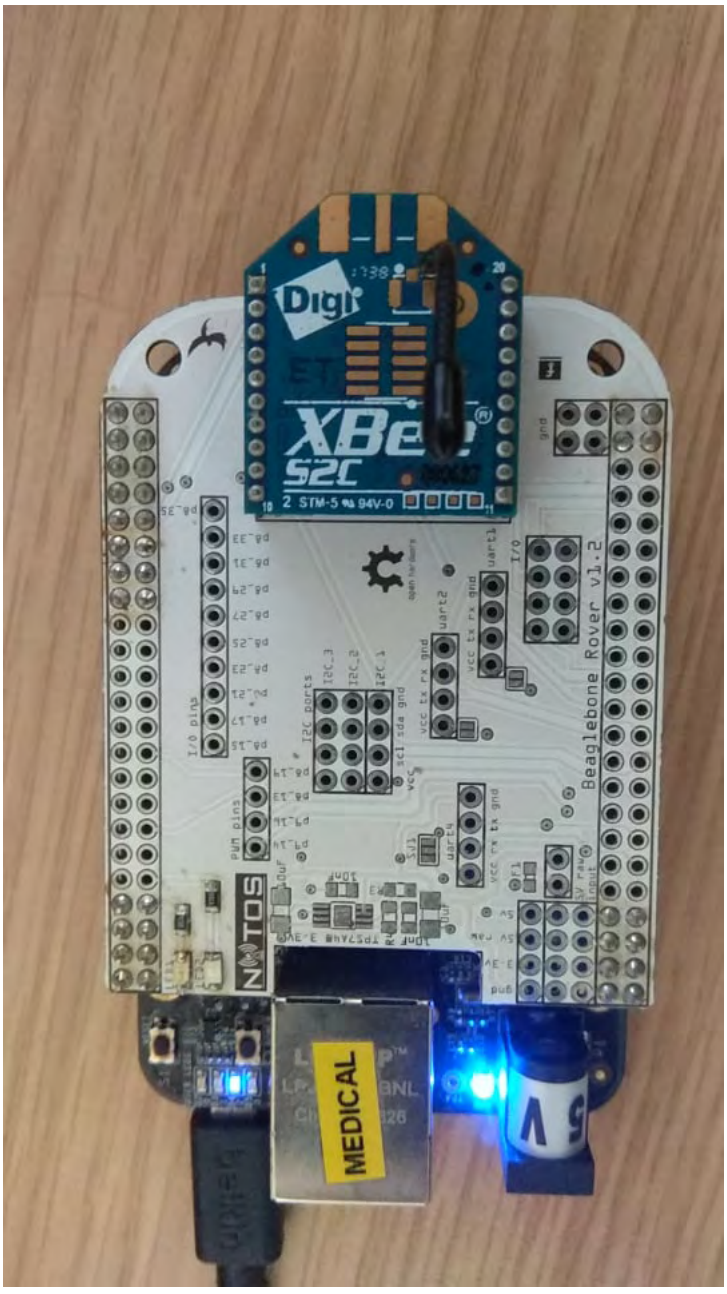

FIGURE 7.2: Gateway.

The gateway in our system is responsible for distributing the necessary information

about the registered clients. It can be connected to a personal computer by USB cable in order to upload the data to the influx database. The connection can also be managed by ethernet cable.

An example of the file that contains the information about the registered clients is depicted below:

7.3 Activation Phase

This phase occurs in the activation of a sniffer node and is continued until the node owns all the necessary information about the registered clients. Once a node gets activated, it waits for an amount of time of seven seconds multiplied by its node ID. This feature is used in order to avoid congestion while transmitting the information of the clients. Then, the node sends a packet with data bytes 0xAA 0xCC to the XBee end device that is connected to. Afterwards, the end device transmits that packet to the XBee coordinator, the beaglebone gets the packet from the XBee coordinator. Then, the beaglebone starts transmitting the packets containing the information of the clients. The information includes the MAC address and the MAC ID of each client. The ESP will receive those packets through the Zigbee network. If all packets are not received in the time interval of ten seconds, then the ESP will send a 0xAA 0xCC packet again. Once all information has been gathered, the ESP will wait for an interval of one minute minus the time they initially waited. This feature is developed in order to synchronize the nodes. Then, the ESP will enter in promiscuous mode. The operational phase begins.

7.4 Operational Phase

The ESP will now start sniffing packets of the Wifi network. If a packet with sender MAC address matches a registered MAC address, then the ESP will measure the RSSI of the packet. It will register the value to a variable and it will consider that MAC address as active. A client will be considered active for two minutes. If two minutes pass without any other packet being sniffed by the client, the client will be considered as inactive. Every ten seconds, the ESP will transmit packets that include the MAC IDs and the latest RSSI values of the active clients. The beaglebone will get those packets by the Zigbee network. Based on each MAC ID, it will access the array with the MAC addresses. Then it will post the MAC address, the name and the RSSI value to the influx database for each received MAC ID. Then, the web graphical user interface will update, activating functions that access the database and parse the data searching for the minimum RSSI value of each client. For each active client, the minimum RSSI value indicates the node that the client is closer to. Thus, the room that the client is in. The graphical user interface will then distribute alphabetically the names of active clients to columns. Each column represents a room

Chapter 8

Testing

8.1 Overview

In this chapter, we will demonstrate our implementation in practice. We tested the implementation with two devices and checked the localization ratio in several cases. The test was conducted on the third floor of network implementation testbed laboratory. We used six nodes. Each node was installed in one of the six big rooms that the third floor disposes. In the following paragraphs, we will present the results of the test.

8.2 Factors that Affect RSSI

It is critical to state that the results and the measurements that our system depends on, come from received signal strength indications. Thus, the results will be affected by the factors that impact the RSSI values. Such factors are:

- Transmitter Power
- Receiver Power
- Orientation of Antennas
- Physical Objects
- Radio Interference from the Wifi networks
- Radio Interference from non Wifi networks
- Distance between Transmitter and Receiver

8.3 Test Results

In the following figure, the third floor of the lab is depicted. The numbered black circles show the number of the room. Respectively, that number is also the node ID of the sniffer deployed in that room. The blue circles represent the position that the sniffers were placed during the test.

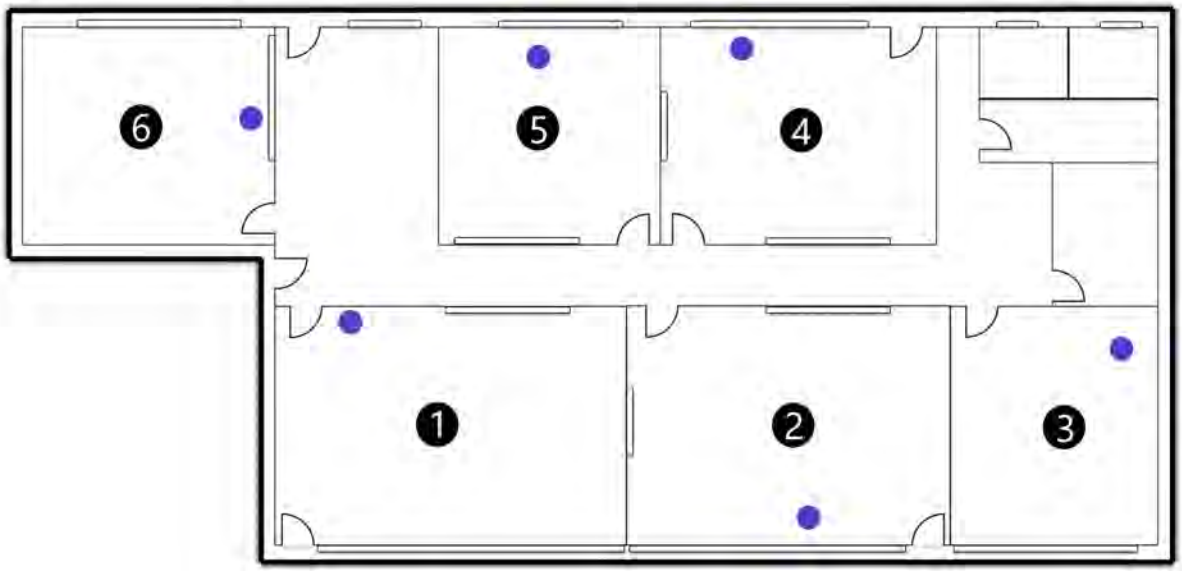

FIGURE 8.1: Third Flood of Nitlab.

At this point, we will represent the measurements for each room. We took an instant RSSI measurement for each spot. We also took an RSSI measurement after one minute of idleness of each device. The measurements are in hexadecimal form. The green marks in the following figures, depict the spots in which we took the measurements. For each picture of a room two tables will be also depicted showing the measurements. The red lines of the tables show the measurements in which the system classified the device in wrong room. We took the measurements with two devices. A Xiaomi Redmi Note 4 and a Samsung Galaxy Tab 2.0.

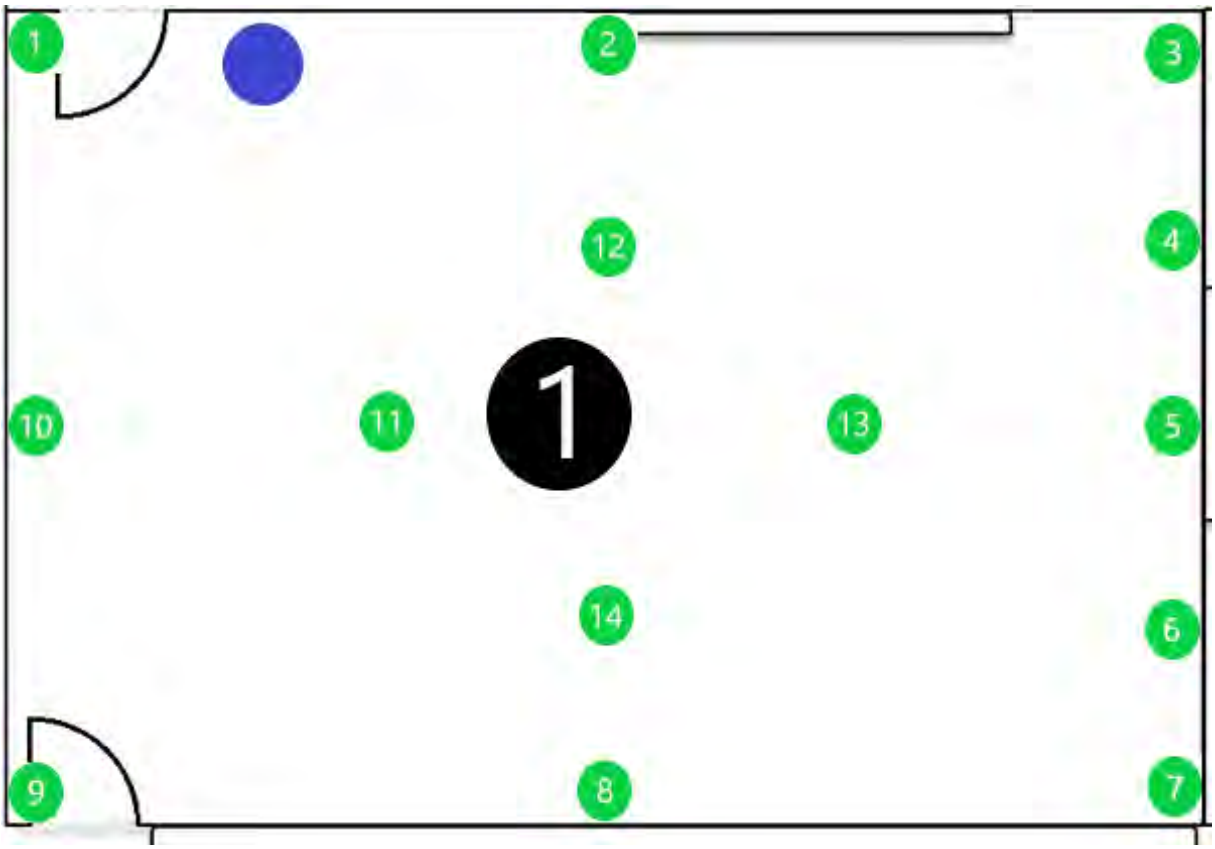

FIGURE 8.2: Room1.

For the Xiaomi Redmi Note 4 the measurements are:

Spot	Node1	Node2	Node3	Node4	Node5	Node6
Spot 1 Measurement 1	2e	3f	3c	>45	>45	43
Spot 1 Measurement 2	2d	3f	3d	>45	>45	41
Spot 2 Measurement 1	23	3f	41	44	3b	3b
Spot 2 Measurement 2	21	3f	42	45	3c	3b
Spot 3 Measurement 1	2b	3e	41	3d	42	43
Spot 3 Measurement 2	2a	3e	40	3d	44	43
Spot 4 Measurement 1	32	34	42	45	41	44
Spot 4 Measurement 2	30	34	42	45	41	>45
Spot 5 Measurement 1	33	34	42	44	44	>45
Spot 5 Measurement 2	31	35	42	>45	41	>45
Spot 6 Measurement 1	35	38	41	>45	3c	43
Spot 6 Measurement 2	35	38	41	>45	3e	44
Spot 7 Measurement 1	37	39	3c	>45	>45	>45
Spot 7 Measurement 2	38	40	3e	>45	>45	>45
Spot 8 Measurement 1	36	43	>45	>45	>45	>45
Spot 8 Measurement 2	35	43	>45	>45	>45	>45
Spot 9 Measurement 1	3a	41	41	44	>45	>45
Spot 9 Measurement 2	3a	42	41	43	>45	>45
Spot 10 Measurement 1	2b	3e	45	42	3e	42
Spot 10 Measurement 2	2c	3e	44	42	3e	41
Spot 11 Measurement 1	31	43	>45	>45	>45	>45
Spot 11 Measurement 2	31	42	>45	>45	>45	>45
Spot 12 Measurement 1	2e	3c	42	>45	45	41
Spot 12 Measurement 2	2c	3c	43	>45	44	41
Spot 13 Measurement 1	32	3a	3b	43	42	>45
Spot 13 Measurement 2	34	3a	3c	44	43	>45
Spot 14 Measurement 1	36	3d	40	45	>45	>45
Spot 14 Measurement 2	37	3e	43	45	>45	>45

For the Samsung Galaxy Tab 2.0 the measurements are:

Spot	Node1	Node2	Node3	Node4	Node5	Node6
Spot 1 Measurement 1	3e	43	3f	>45	>45	>45
Spot 1 Measurement 2	3c	41	3f	>45	>45	>45
Spot 2 Measurement 1	2e	3e	40	>45	>45	>45
Spot 2 Measurement 2	2e	3d	41	>45	>45	>45
Spot 3 Measurement 1	37	42	>45	>45	43	44
Spot 3 Measurement 2	3a	42	>45	>45	44	>45
Spot 4 Measurement 1	39	3b	44	>45	42	41
Spot 4 Measurement 2	3f	3b	43	>45	41	<45
Spot 5 Measurement 1	35	40	>45	>45	44	>45
Spot 5 Measurement 2	38	40	>45	>45	45	>45
Spot 6 Measurement 1	34	41	>45	>45	45	>45
Spot 6 Measurement 2	35	41	>45	>45	>45	>45
Spot 7 Measurement 1	3d	40	>45	>45	43	44
Spot 7 Measurement 2	3c	38	>45	>45	43	44
Spot 8 Measurement 1	2e	>45	>45	>45	>45	3e
Spot 8 Measurement 2	2e	>45	>45	>45	>45	42
Spot 9 Measurement 1	2d	40	45	>45	>45	>45
Spot 9 Measurement 2	2f	40	>45	>45	>45	>45
Spot 10 Measurement 1	2b	43	>45	>45	>45	40
Spot 10 Measurement 2	2b	41	>45	>45	>45	41
Spot 11 Measurement 1	2a	42	>45	>45	43	44
Spot 11 Measurement 2	2b	40	>45	>45	43	43
Spot 12 Measurement 1	2d	39	>45	>45	44	43
Spot 12 Measurement 2	2d	38	>45	>45	43	43
Spot 13 Measurement 1	35	3d	>45	>45	>45	>45
Spot 13 Measurement 2	35	3d	>45	>45	>45	>45
Spot 14 Measurement 1	36	41	42	>45	>45	>45
Spot 14 Measurement 2	37	41	43	>45	>45	>45

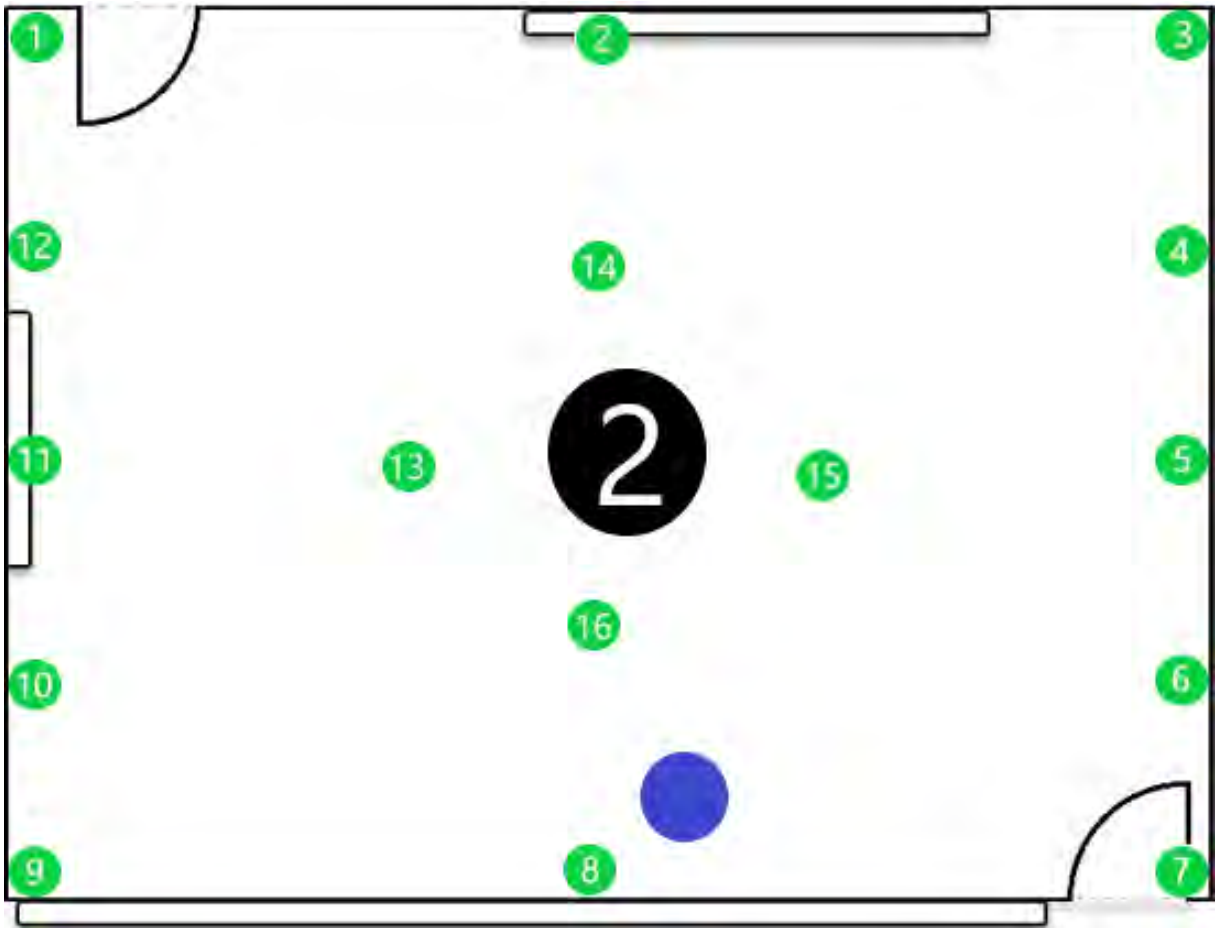

FIGURE 8.3: Room2.

For the Xiaomi Redmi Note 4 the measurements are:

Spot	Node1	Node2	Node3	Node4	Node5	Node6
Spot 1 Measurement 1	2d	2e	35	>45	3c	41
Spot 1 Measurement 2	2f	3a	3d	>45	3d	40
Spot 2 Measurement 1	3b	2f	36	44	42	45
Spot 2 Measurement 2	3e	2c	3a	44	>45	>45
Spot 3 Measurement 1	3a	37	38	>45	41	>45
Spot 3 Measurement 2	3a	38	35	>45	44	>45
Spot 4 Measurement 1	42	2b	39	3e	44	44
Spot 4 Measurement 2	42	2c	39	3e	45	>45
Spot 5 Measurement 1	3b	35	3a	>45	>45	>45
Spot 5 Measurement 2	3b	3d	35	>45	3f	43
Spot 6 Measurement 1	44	2b	40	3e	>45	>45
Spot 6 Measurement 2	44	2e	43	3f	>45	>45
Spot 7 Measurement 1	39	2d	32	3d	43	42
Spot 7 Measurement 2	38	2e	35	3f	43	44
Spot 8 Measurement 1	3d	31	3f	>45	>45	43
Spot 8 Measurement 2	3e	41	36	>45	>45	43
Spot 9 Measurement 1	33	29	36	44	>45	45
Spot 9 Measurement 2	37	30	39	45	>45	>45
Spot 10 Measurement 1	34	30	37	45	>45	>45
Spot 10 Measurement 2	37	31	38	42	>45	>45
Spot 11 Measurement 1	33	2c	37	42	43	40
Spot 11 Measurement 2	31	2d	35	>45	>45	41
Spot 12 Measurement 1	2f	2d	38	40	40	44
Spot 12 Measurement 2	2f	30	38	44	>45	43
Spot 13 Measurement 1	34	27	39	3a	43	3c
Spot 13 Measurement 2	34	29	44	3d	43	3f
Spot 14 Measurement 1	36	3a	43	3c	42	44
Spot 14 Measurement 2	38	28	44	3b	41	>45
Spot 15 Measurement 1	3d	34	35	43	43	43
Spot 15 Measurement 2	3d	35	36	44	43	43
Spot 16 Measurement 1	34	27	39	3a	43	3c
Spot 16 Measurement 2	35	28	39	29	45	3e

For the Samsung Galaxy Tab 2.0 the measurements are:

Spot	Node1	Node2	Node3	Node4	Node5	Node6
Spot 1 Measurement 1	3c	36	3f	45	>45	>45
Spot 1 Measurement 2	3c	34	3f	>45	>45	>45
Spot 2 Measurement 1	3b	35	3f	43	45	>45
Spot 2 Measurement 2	3b	33	36	>45	>45	>45
Spot 3 Measurement 1	3b	30	33	41	45	>45
Spot 3 Measurement 2	3c	31	33	40	44	>45
Spot 4 Measurement 1	38	35	39	42	>45	>45
Spot 4 Measurement 2	40	33	38	41	>45	>45
Spot 5 Measurement 1	41	34	37	41	44	>45
Spot 5 Measurement 2	42	34	33	41	40	>45
Spot 6 Measurement 1	3a	35	3f	3d	45	>45
Spot 6 Measurement 2	3a	34	3d	41	>45	>45
Spot 7 Measurement 1	3c	3b	3f	3c	44	>45
Spot 7 Measurement 2	3e	39	3d	40	43	>45
Spot 8 Measurement 1	32	2b	42	>45	>45	>45
Spot 8 Measurement 2	33	2c	44	>45	>45	>45
Spot 9 Measurement 1	38	37	3a	>45	44	>45
Spot 9 Measurement 2	36	38	3b	>45	>45	>45
Spot 10 Measurement 1	38	36	3c	>45	>45	>45
Spot 10 Measurement 2	38	36	3f	>45	>45	>45
Spot 11 Measurement 1	35	3b	41	40	>45	>45
Spot 11 Measurement 2	38	3b	41	42	>45	>45
Spot 12 Measurement 1	36	3e	3c	40	>45	>45
Spot 12 Measurement 2	38	3b	3d	42	>45	>45
Spot 13 Measurement 1	36	37	3e	43	40	>45
Spot 13 Measurement 2	38	38	3d	42	44	>45
Spot 14 Measurement 1	39	2f	42	41	>45	>45
Spot 14 Measurement 2	37	2f	43	44	>45	>45
Spot 15 Measurement 1	>45	34	41	>45	44	>45
Spot 15 Measurement 2	43	32	43	>45	>45	>45
Spot 16 Measurement 1	40	30	45	>45	>45	>45
Spot 16 Measurement 2	37	30	43	>45	>45	>45

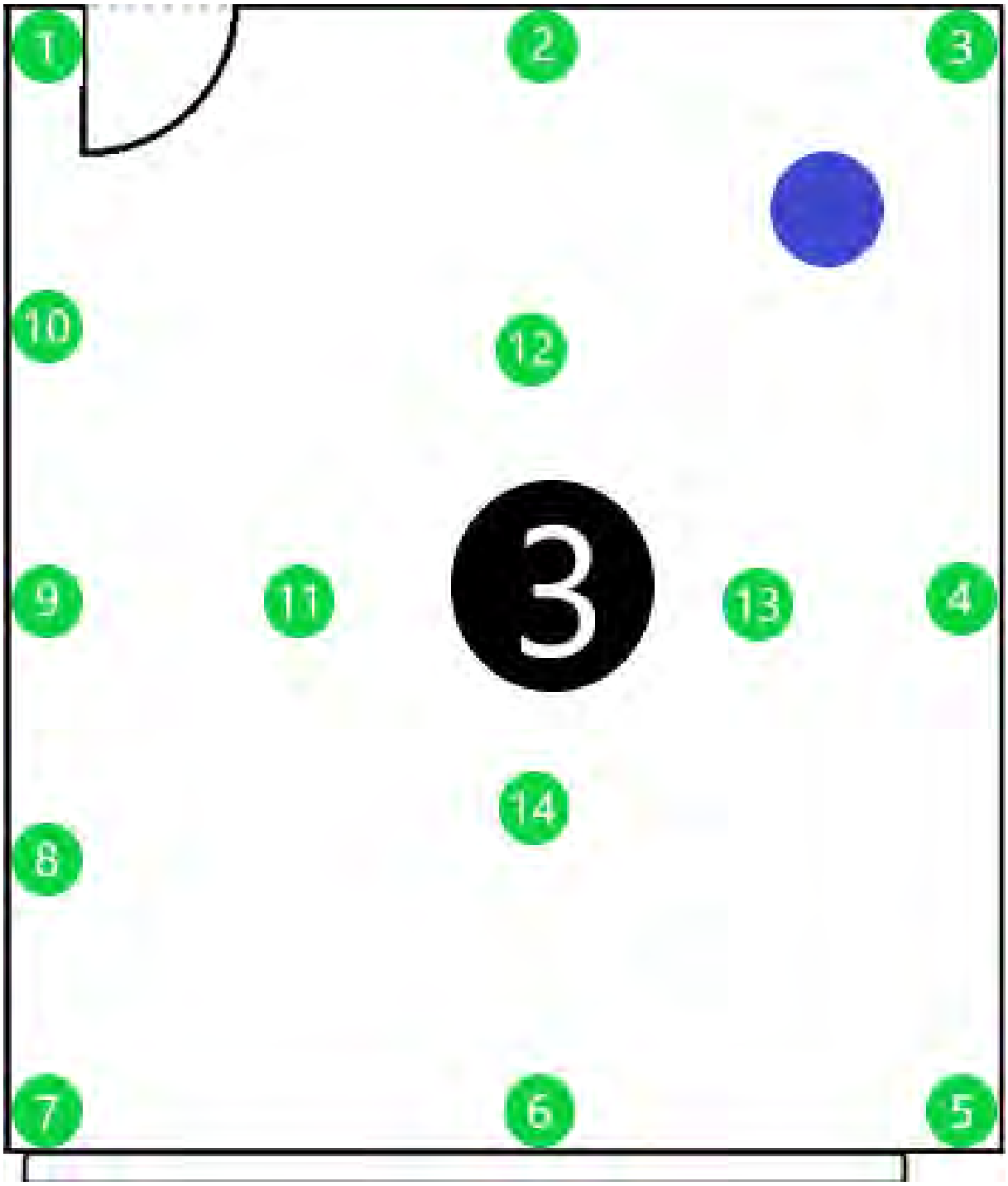

FIGURE 8.4: Room3.

For the Xiaomi Redmi Note 4 the measurements are:

Spot	Node1	Node2	Node3	Node4	Node5	Node6
Spot 1 Measurement 1	3c	32	35	>45	>45	43
Spot 1 Measurement 2	3e	3d	32	>45	>45	>45
Spot 2 Measurement 1	40	37	38	>45	>45	>45
Spot 2 Measurement 2	37	36	33	>45	>45	>45
Spot 3 Measurement 1	41	3c	37	>45	>45	>45
Spot 3 Measurement 1	43	3b	39	>45	>45	>45
Spot 4 Measurement 1	43	44	2d	3f	>45	>45
Spot 4 Measurement 2	43	44	2e	3d	>45	>45
Spot 5 Measurement 1	>45	44	30	3f	>45	>45
Spot 5 Measurement 2	>45	44	30	3e	>45	>45
Spot 6 Measurement 1	42	40	32	>45	43	>45
Spot 6 Measurement 2	44	43	34	>45	>45	>45
Spot 7 Measurement 1	42	41	37	44	43	>45
Spot 7 Measurement 2	42	41	36	42	43	>45
Spot 8 Measurement 1	43	41	32	44	41	>45
Spot 8 Measurement 2	44	41	33	>45	44	>45
Spot 9 Measurement 1	43	44	30	43	>45	>45
Spot 9 Measurement 2	43	44	33	44	>45	>45
Spot 10 Measurement 1	3e	40	38	>45	42	>45
Spot 10 Measurement 2	3f	41	37	>45	>45	>45
Spot 11 Measurement 1	3f	37	2e	>45	>45	>45
Spot 11 Measurement 2	3e	37	2f	>45	>45	>45
Spot 12 Measurement 1	3a	34	30	>45	>45	44
Spot 12 Measurement 2	3a	32	31	>45	>45	>45
Spot 13 Measurement 1	>45	34	2e	44	>45	>45
Spot 13 Measurement 2	>45	34	2f	43	>45	>45
Spot 14 Measurement 1	>45	3e	26	>45	>45	>45
Spot 14 Measurement 2	>45	3d	28	>45	>45	>45

For the Samsung Galaxy Tab 2.0 the measurements are:

Spot	Node1	Node2	Node3	Node4	Node5	Node6
Spot 1 Measurement 1	3b	41	31	45	>45	>45
Spot 1 Measurement 2	3c	3f	30	>45	>45	>45
Spot 2 Measurement 1	41	3e	31	>45	>45	>45
Spot 2 Measurement 2	41	3d	30	>45	>45	>45
Spot 3 Measurement 1	>45	3e	3c	43	>45	>45
Spot 3 Measurement 2	>45	3f	3a	43	>45	>45
Spot 4 Measurement 1	3d	42	2d	>45	>45	>45
Spot 4 Measurement 2	3f	42	2d	>45	>45	>45
Spot 5 Measurement 1	41	3f	30	>45	>45	>45
Spot 5 Measurement 2	41	3f	34	>45	>45	>45
Spot 6 Measurement 1	41	41	2d	>45	>45	>45
Spot 6 Measurement 2	44	41	2e	>45	>45	>45
Spot 7 Measurement 1	3d	3f	33	>45	>45	>45
Spot 7 Measurement 2	3d	3c	34	>45	>45	>45
Spot 8 Measurement 1	43	3d	2f	>45	>45	>45
Spot 8 Measurement 2	43	3f	2d	>45	>45	>45
Spot 9 Measurement 1	42	3c	2b	>45	>45	>45
Spot 9 Measurement 2	44	3b	2c	>45	>45	>45
Spot 10 Measurement 1	43	43	2f	>45	>45	>45
Spot 10 Measurement 2	44	43	2f	>45	>45	>45
Spot 11 Measurement 1	3b	3c	34	>45	>45	>45
Spot 11 Measurement 2	3a	3d	36	>45	>45	>45
Spot 12 Measurement 1	3d	42	33	42	>45	>45
Spot 12 Measurement 2	3f	40	35	40	>45	>45
Spot 13 Measurement 1	>45	44	3f	>45	>45	>45
Spot 13 Measurement 2	>45	44	3c	>45	>45	>45
Spot 14 Measurement 1	3b	40	36	44	44	>45
Spot 14 Measurement 2	3f	41	33	42	>45	>45

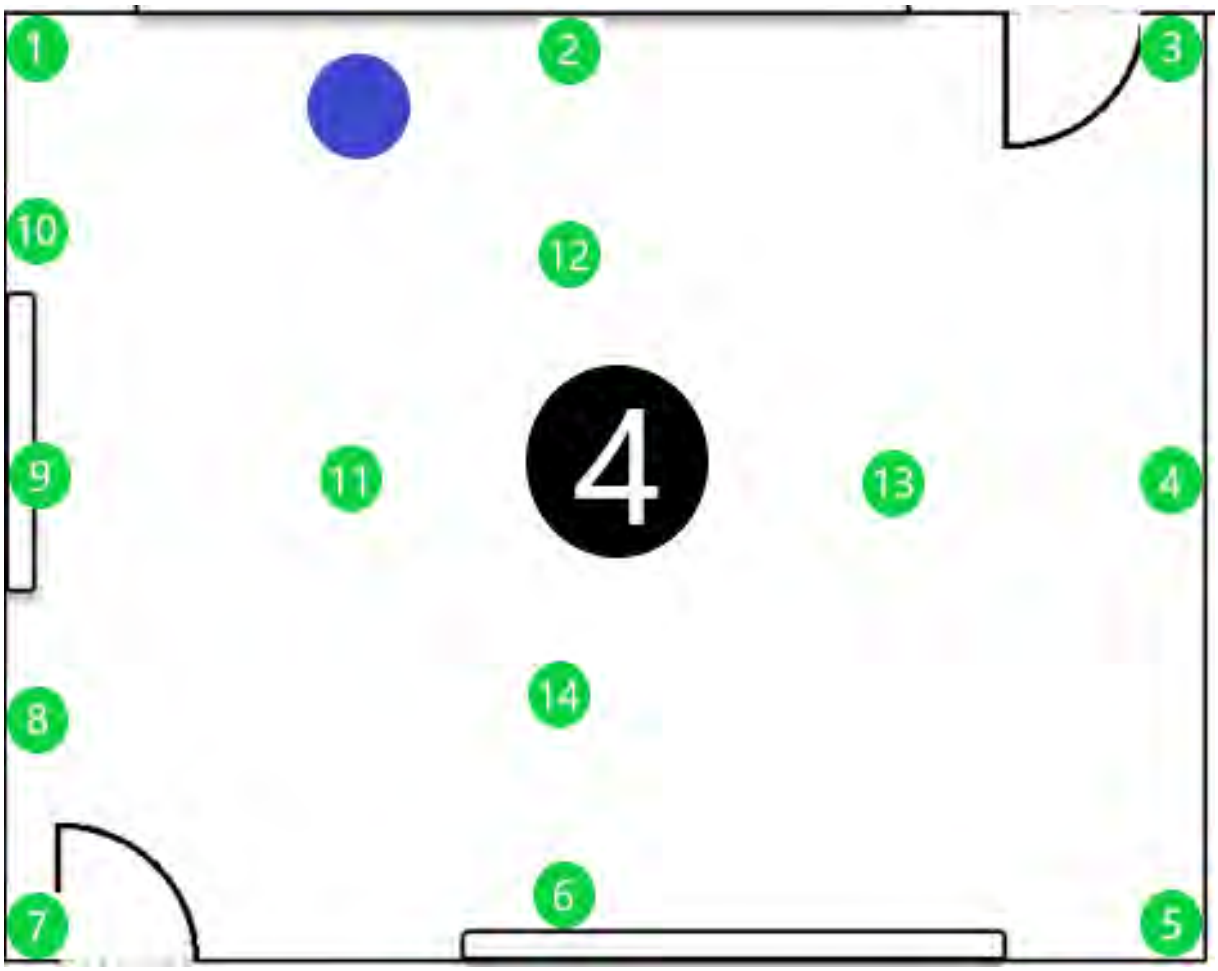

FIGURE 8.5: Room4.

For the Xiaomi Redmi Note 4 the measurements are:

Spot	Node1	Node2	Node3	Node4	Node5	Node6
Spot 1 Measurement 1	>45	41	44	2f	45	>45
Spot 1 Measurement 2	>45	43	40	2e	45	>45
Spot 2 Measurement 1	>45	>45	42	2d	>45	>45
Spot 2 Measurement 2	>45	>45	43	2f	>45	>45
Spot 3 Measurement 1	43	43	>45	30	43	>45
Spot 3 Measurement 1	>45	43	>45	2e	40	>45
Spot 4 Measurement 1	42	42	3c	35	3c	42
Spot 4 Measurement 2	>45	44	3d	33	3f	>45
Spot 5 Measurement 1	>45	41	45	30	45	>45
Spot 5 Measurement 2	>45	43	44	2f	>45	>45
Spot 6 Measurement 1	>45	3c	>45	38	36	3f
Spot 6 Measurement 2	>45	3d	>45	39	35	41
Spot 7 Measurement 1	>45	3a	>45	36	33	42
Spot 7 Measurement 2	>45	3c	>45	36	32	42
Spot 8 Measurement 1	43	44	>45	32	38	3f
Spot 8 Measurement 2	44	44	>45	33	32	41
Spot 9 Measurement 1	42	43	>45	31	37	41
Spot 9 Measurement 2	42	44	>45	31	35	42
Spot 10 Measurement 1	>45	>45	>45	34	2f	>45
Spot 10 Measurement 2	>45	>45	>45	35	2c	>45
Spot 11 Measurement 1	43	44	42	31	>45	>45
Spot 11 Measurement 2	>45	45	42	33	>45	>45
Spot 12 Measurement 1	>45	>45	>45	2d	3f	>45
Spot 12 Measurement 2	>45	>45	>45	2f	41	>45
Spot 13 Measurement 1	41	38	3c	2e	3b	3e
Spot 13 Measurement 2	41	41	3f	2f	3b	3f
Spot 14 Measurement 1	>45	43	3d	34	45	>45
Spot 14 Measurement 2	>45	43	3c	35	>45	>45

For the Samsung Galaxy Tab 2.0 the measurements are:

Spot	Node1	Node2	Node3	Node4	Node5	Node6
Spot 1 Measurement 1	>45	43	>45	31	40	>45
Spot 1 Measurement 2	>45	44	>45	33	40	>45
Spot 2 Measurement 1	>45	>45	>45	30	43	>45
Spot 2 Measurement 2	>45	>45	>45	30	>45	>45
Spot 3 Measurement 1	>45	41	44	2f	45	>45
Spot 3 Measurement 2	>45	42	42	2d	>45	>45
Spot 4 Measurement 1	43	45	41	3d	>45	>45
Spot 4 Measurement 2	>45	44	40	3d	>45	>45
Spot 5 Measurement 1	45	40	>45	30	3d	>45
Spot 5 Measurement 2	>45	41	>45	33	3f	>45
Spot 6 Measurement 1	44	38	>45	35	3c	>45
Spot 6 Measurement 2	>45	39	>45	35	3e	>45
Spot 7 Measurement 1	45	3f	>45	38	3a	>45
Spot 7 Measurement 2	44	41	>45	36	3a	>45
Spot 8 Measurement 1	44	>45	45	31	39	>45
Spot 8 Measurement 2	43	>45	>45	34	39	>45
Spot 9 Measurement 1	>45	>45	>45	32	37	>45
Spot 9 Measurement 2	>45	>45	>45	33	35	>45
Spot 10 Measurement 1	>45	>45	>45	2f	34	>45
Spot 10 Measurement 2	>45	>45	>45	31	36	>45
Spot 11 Measurement 1	43	45	>45	3d	>45	>45
Spot 11 Measurement 2	>45	44	>45	3b	>45	>45
Spot 12 Measurement 1	>45	>45	>45	38	41	>45
Spot 12 Measurement 2	>45	>45	>45	38	43	>45
Spot 13 Measurement 1	43	44	41	3b	44	>45
Spot 13 Measurement 2	>45	>45	42	3b	>45	>45
Spot 14 Measurement 1	>45	>45	44	3d	>45	>45
Spot 14 Measurement 2	>45	>45	45	3c	>45	>45

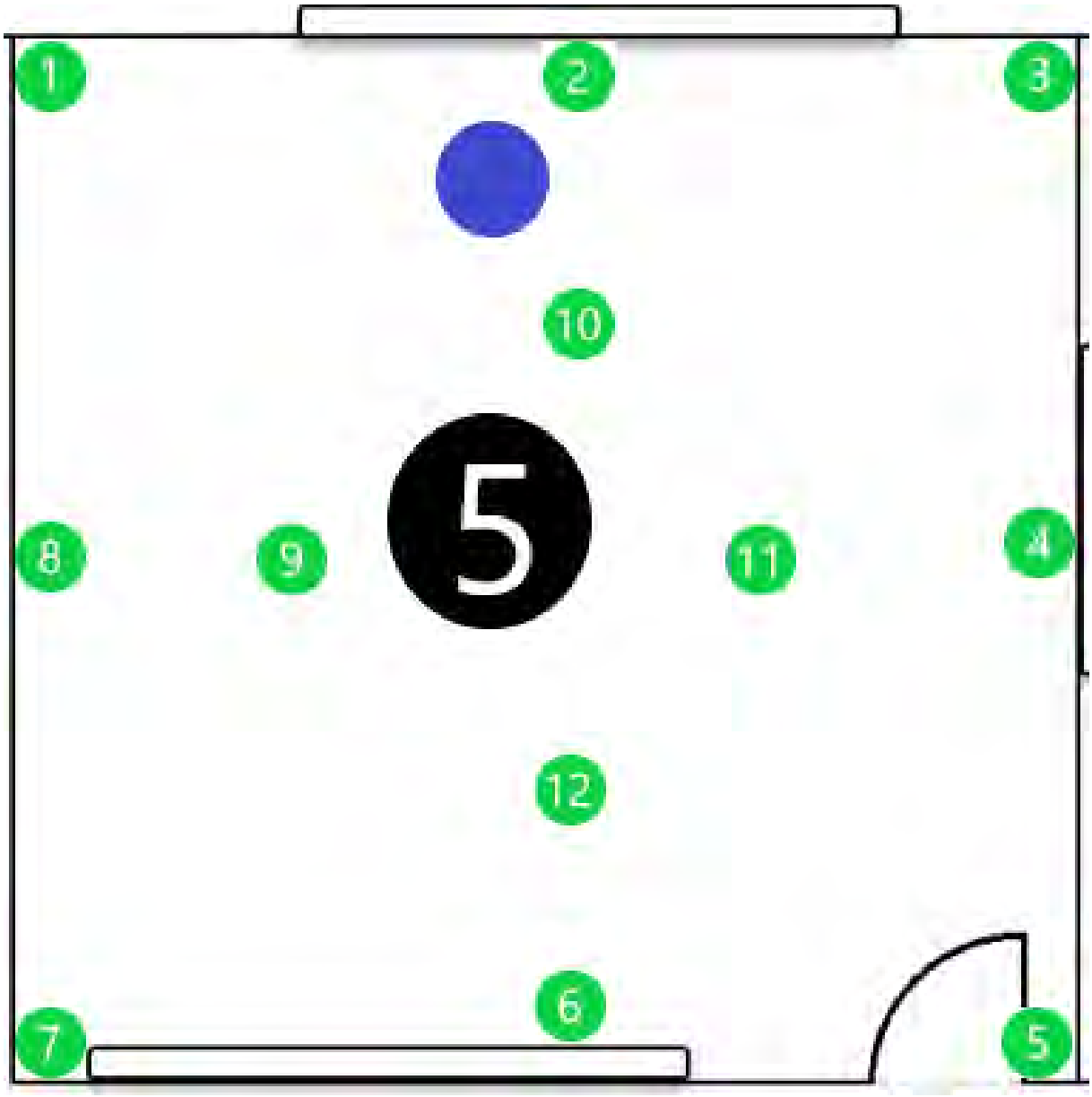

FIGURE 8.6: Room5.

For the Xiaomi Redmi Note 4 the measurements are:

Spot	Node1	Node2	Node3	Node4	Node5	Node6
Spot 1 Measurement 1	>45	>45	>45	43	2e	3f
Spot 1 Measurement 2	>45	>45	>45	44	2f	3d
Spot 2 Measurement 1	>45	44	>45	45	25	44
Spot 2 Measurement 2	>45	>45	>45	44	23	>45
Spot 3 Measurement 1	>45	>45	>45	38	3a	>45
Spot 3 Measurement 2	>45	>45	>45	39	3a	>45
Spot 4 Measurement 1	>45	41	44	3f	38	42
Spot 4 Measurement 2	>45	44	>45	3e	3a	44
Spot 5 Measurement 1	40	41	>45	3d	3f	>45
Spot 5 Measurement 2	43	44	>45	3c	3f	>45
Spot 6 Measurement 1	44	43	>45	>45	36	3f
Spot 6 Measurement 2	>45	44	>45	>45	37	41
Spot 7 Measurement 1	3f	3d	>45	44	3c	40
Spot 7 Measurement 2	40	40	>45	43	3f	42
Spot 8 Measurement 1	3b	3a	>45	3e	39	34
Spot 8 Measurement 2	3a	42	>45	43	37	3b
Spot 9 Measurement 1	3f	3d	>45	40	37	35
Spot 9 Measurement 2	43	42	>45	3e	38	37
Spot 10 Measurement 1	40	3a	>45	45	29	42
Spot 10 Measurement 2	41	3c	>45	44	2c	43
Spot 11 Measurement 1	3e	3d	>45	43	36	3c
Spot 11 Measurement 2	41	3c	>45	44	36	3b
Spot 12 Measurement 1	40	3a	>45	45	29	42
Spot 12 Measurement 2	41	3d	>45	42	2b	>45

For the Samsung Galaxy Tab 2.0 the measurements are:

Spot	Node1	Node2	Node3	Node4	Node5	Node6
Spot 1 Measurement 1	>45	>45	>45	44	31	40
Spot 1 Measurement 2	>45	>45	>45	41	33	43
Spot 2 Measurement 1	>45	>45	>45	42	28	>45
Spot 2 Measurement 2	>45	>45	>45	41	2a	>45
Spot 3 Measurement 1	>45	>45	>45	3b	3d	>45
Spot 3 Measurement 2	>45	>45	>45	39	3a	>45
Spot 4 Measurement 1	43	45	41	3d	3a	>45
Spot 4 Measurement 2	>45	44	40	3d	39	>45
Spot 5 Measurement 1	43	42	>45	3e	40	>45
Spot 5 Measurement 2	44	41	>45	3e	3d	>45
Spot 6 Measurement 1	44	42	44	43	3c	>45
Spot 6 Measurement 2	>45	39	42	44	3e	>45
Spot 7 Measurement 1	40	3f	>45	44	3d	41
Spot 7 Measurement 2	44	41	>45	42	3d	43
Spot 8 Measurement 1	39	3f	43	3d	40	>45
Spot 8 Measurement 2	40	3e	41	3d	40	>45
Spot 9 Measurement 1	43	3f	44	3c	3e	>45
Spot 9 Measurement 2	40	3c	41	3c	3f	>45
Spot 10 Measurement 1	40	3c	41	3c	3f	>45
Spot 10 Measurement 2	3e	40	>45	3c	3d	40
Spot 11 Measurement 1	3f	42	>45	3c	3c	3d
Spot 11 Measurement 2	>45	>45	>45	34	39	>45
Spot 12 Measurement 1	3e	>45	>45	3d	3d	42
Spot 12 Measurement 2	3f	>45	>45	3f	3c	44

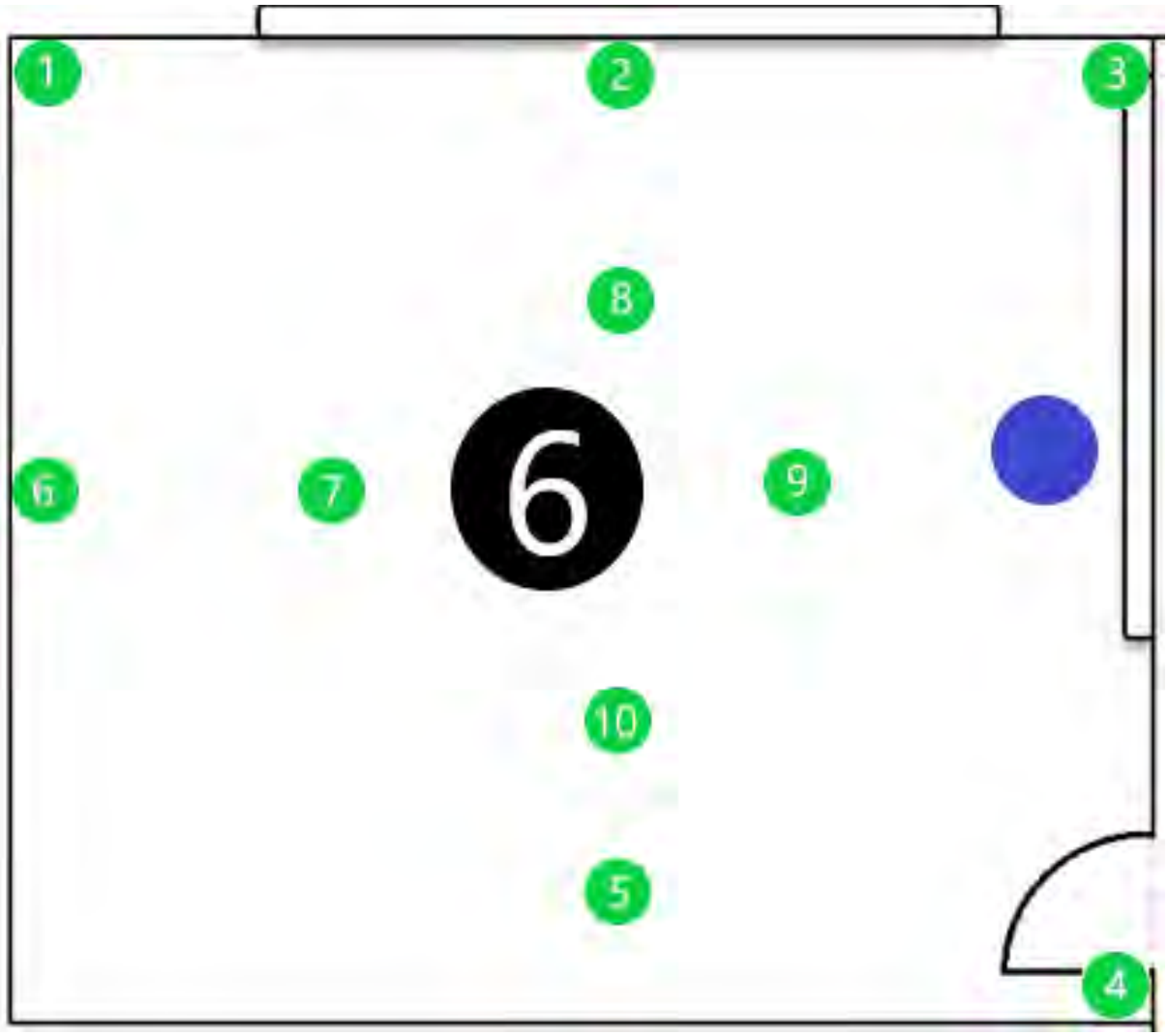

FIGURE 8.7: Room6.

For the Xiaomi Redmi Note 4 the measurements are:

Spot	Node1	Node2	Node3	Node4	Node5	Node6
Spot 1 Measurement 1	>45	>45	42	>45	>45	3a
Spot 1 Measurement 2	>45	>45	>45	>45	>45	2f
Spot 2 Measurement 1	44	>45	>45	44	>45	32
Spot 2 Measurement 2	>45	>45	>45	44	>45	30
Spot 3 Measurement 1	44	>45	>45	>45	42	3b
Spot 3 Measurement 2	>45	>45	>45	>45	42	3d
Spot 4 Measurement 1	43	>45	>45	>45	3f	3b
Spot 4 Measurement 2	44	44	>45	>45	3d	3b
Spot 5 Measurement 1	>45	>45	>45	>45	39	35
Spot 5 Measurement 2	43	>45	>45	>45	3c	34
Spot 6 Measurement 1	>45	>45	>45	>45	42	37
Spot 6 Measurement 2	>45	>45	>45	>45	44	38
Spot 7 Measurement 1	40	45	>45	>45	>45	36
Spot 7 Measurement 2	3f	45	>45	44	>45	36
Spot 8 Measurement 1	>45	>45	>45	>45	44	39
Spot 8 Measurement 2	44	45	>45	>45	43	36
Spot 9 Measurement 1	3c	3c	>45	>45	41	2b
Spot 9 Measurement 2	3d	40	>45	>45	44	2e
Spot 10 Measurement 1	>45	>45	>45	>45	>45	36
Spot 10 Measurement 2	41	44	>45	>45	>45	35

For the Samsung Galaxy Tab 2.0 the measurements are:

Spot	Node1	Node2	Node3	Node4	Node5	Node6
Spot 1 Measurement 1	>45	>45	>45	>45	>45	3b
Spot 1 Measurement 2	>45	>45	>45	>45	>45	2e
Spot 2 Measurement 1	>45	>45	>45	>45	>45	32
Spot 2 Measurement 2	>45	>45	>45	>45	>45	32
Spot 3 Measurement 1	>45	>45	>45	>45	>45	3b
Spot 3 Measurement 2	>45	>45	>45	>45	45	3e
Spot 4 Measurement 1	>45	>45	>45	>45	41	3e
Spot 4 Measurement 2	>45	>45	>45	>45	40	3c
Spot 5 Measurement 1	>45	>45	>45	>45	38	35
Spot 5 Measurement 2	>45	>45	>45	>45	3f	33
Spot 6 Measurement 1	>45	>45	>45	>45	43	38
Spot 6 Measurement 2	>45	>45	>45	>45	44	3a
Spot 7 Measurement 1	42	>45	>45	>45	>45	3c
Spot 7 Measurement 2	43	>45	>45	>45	>45	3a
Spot 8 Measurement 1	>45	>45	>45	>45	44	3c
Spot 8 Measurement 2	>45	>45	>45	>45	43	3a
Spot 9 Measurement 1	40	3e	>45	>45	44	2d
Spot 9 Measurement 2	41	40	>45	>45	43	2e
Spot 10 Measurement 1	41	>45	>45	>45	>45	35
Spot 10 Measurement 2	44	44	>45	>45	>45	38

8.4 Summary

After inspecting the test results, we can note that the system is not one hundred percent successful. There are false localization results especially in room two and five. This is due to the placement of the sniffer nodes. The sniffer nodes were placed at the height of the desks, prone to many obstacles such as computers, people and furniture. In order for the system to be functioning in a more optimal way, the sniffer nodes should be placed more strategically and at a bigger height that guarantees minimum interference with physical obstacles. The system could also be optimized with algorithms such as the k nearest neighbour algorithm instead of just localizing based on the minimum absolute rssi value.

Analytically the percentage of localization success per room:

- Room1 Xiaomi Redmi Note 4: 100% localization success
- Room1 Samsung Galaxy Tab 2.0: 96% localization success
- Room2 Xiaomi Redmi Note 4: 84% localization success
- Room2 Samsung Galaxy Tab 2.0: 75% localization success
- Room3 Xiaomi Redmi Note 4: 92% localization success

- Room3 Samsung Galaxy Tab 2.0: 100% localization success
- Room4 Xiaomi Redmi Note 4: 96% localization success
- Room4 Samsung Galaxy Tab 2.0: 100% localization success
- Room5 Xiaomi Redmi Note 4: 70% localization success
- Room5 Samsung Galaxy Tab 2.0: 50% localization success
- Room6 Xiaomi Redmi Note 4: 100% localization success
- Room6 Samsung Galaxy Tab 2.0: 100% localization success

Chapter 9

Conclusion

In this thesis, we aimed to implement a WiFi localization system. Firstly, we presented the various types and subtypes of WiFi packets. We stated which ones we found useful for the implementation and we analyzed their role in a WiFi network.

Thereafter, we introduced the ESP-32 Devkits, the devices we used as sniffer-nodes. We also demonstrated the role of a sniffer in a WiFi network. We inspected the critical functionalities of the code that the ESPs use.

In the next chapter, we inspected the mediator of our implementation. The XBee devices. We defined what a Zigbee network is and we introduced the features and the configuration that the XBee devices use in our network.

Afterwards, we showed the Beaglebone Black device. We inspected its features and its role as a gateway in our implemenation. We analyzed step by step the way we set it up. We also demonstrated its features through the code that it executes.

Then, we introduced the tool that parses the database where all the useful data for the localization are uploaded. We also demonstrated the Web GUI tool we used in order to show the position of each registered localized client.

Lastly, we deployed the nodes in the third floor of the laboratory. We cited the results of the localization measurements we took. We also demonstrated the percentage of successful localization cases in each room. We conclude that the system can be optimized with a more strategical placement of the sniffer nodes. It can also combine more algorithms instead of exploiting only the minimum absolute RSSI value.

Bibliography

[1] Timea Bagosi and Zoltan Baruch. "Indoor Localization by WiFi". In: (2011).

[2] Balena. https://www.balena.io/etcher/.

[3] BeagleboardOrg. https://beagleboard.org/black.

[4] Core-Electronics. *What are xbee modules*. https://core-electronics.com.au/tutorials/what-are-xbee-modules.html.

[5] DIGI. https://www.digi.com/.

[6] DIGI. *Standards and Technologies*. https://www.digi.com/resources/standards-and-technologies/zigbee-wireless-mesh-networking.

[7] *Esp32 Devkit V1 Documentation*. https://docs.zerynth.com/latest/official/board.zerynth.doit_esp32/docs/index.html.

[8] *Esp32 WROOM 32 Documentation*. https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf.

[9] Matthew S. Gast. *802.11 Wireless Networks: The Definitive Guide, 2nd Edition*.

[10] Beagleboard Latest Images. https://beagleboard.org/latest-images.

[11] *Influxdb Module*. https://pypi.org/project/influxdb/.

[12] *Influxdb Wiki*. https://en.wikipedia.org/wiki/InfluxDB.

[13] Lei Shu Trung Q. Duong Yuanfang Chen Jianwei Niu Bowei Wang. "ZIL: An Energy-Efficient Indoor Localization System Using ZigBee Radio to Detect WiFi Fingerprints". In: *IEEE Journal on Selected Areas in Communications* (2015), pp. 1431–1442.

[14] Beagleboard Getting Started. https://beagleboard.org/getting-started.

[15] *Talking To Beaglebone*. http://shallowsky.com/blog/hardware/talking-to-beaglebone.html.

[16] *UART Definition*. https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter.

[17] *Xbee Devices Module*. https://xbplib.readthedocs.io/en/latest/api/digi.xbee.devices.html.

[18] *XBee S2C Documentation*. https://www.digi.com/resources/documentation/digidocs/pdfs/90002002.pdf.

[19] Yuan Feng Xiuyan Zhu. "RSSI-based Algorithm for Indoor Localization". In: (May 2013).

[20] Rosdiadee Nordin Zahid Farid and Mahamod Ismail. "Recent Advances in Wireless Indoor Localization Techniques and System". In: *Journal of Computer Networks and Communications* (2013).