LO53 Wi-Fi Positioning System
Lab report

Thomas Gagneret
Stéphane Parunakian
Tao Sauvage

Contents

1 Access Points **2**

1.1 Linked list of the UEs 2

1.2 Passive Sniffer 5

1.3 HTTP Daemon 6

1.4 Run the program 7

2 Positioning server **8**

2.1 Tools and versions 8

2.2 Programm 8

2.3 Database 10

2.4 Configuration file 11

3 Android **13**

3.1 Settings 13

3.2 Setup Tool 14

3.3 Positioning 15

3.4 RSSI values 15

Chapter 1

Access Points

The first main component of the Wi-Fi positioning system is the configuration of the Access Points. In this chapter we describe how an AP works and how to configure it.

1.1 Linked list of the UEs

First of all, the AP needs a data structure in order to save the different measures of each User Equipment.

1.1.1 Internal data structure

Since UE measurements can be described as a simple structure containing only two attributes – the RSSI value and when this value has been retrieved – its seemed to be obvious to use a linked list.

The timer gives the AP the ability to delete a sample after a certain amount of time which ensures to only keep accurate new values.

It becomes natural that the list is composed of elements where each of them contains the MAC address of UE and a sample as shown below:

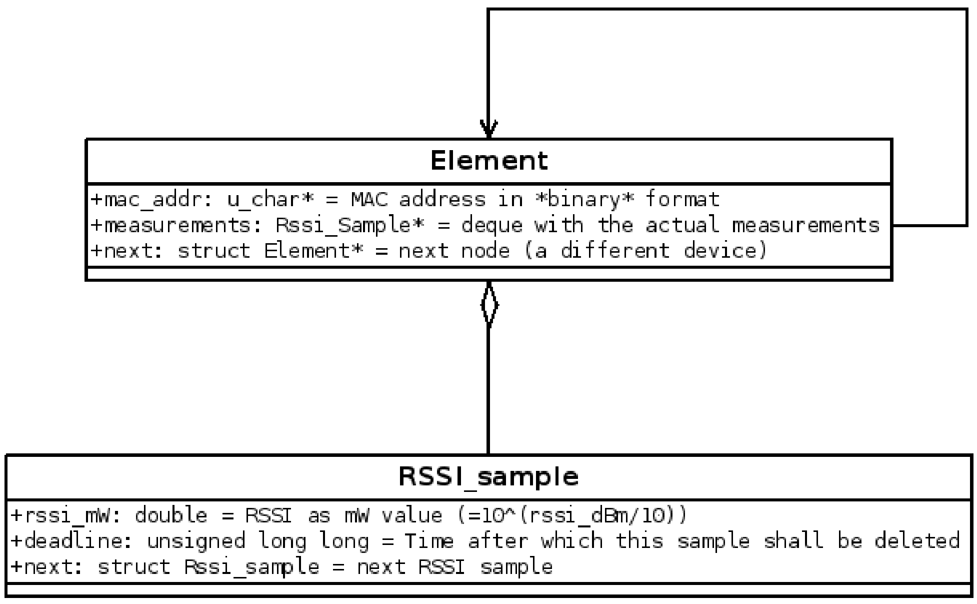

Element
+mac_addr: u_char* = MAC address in *binary* format
+measurements: Rssi_Sample* = deque with the actual measurements
+next: struct Element* = next node (a different device)
RSSI_sample
+rssi_mW: double = RSSI as mW value (=10^(rssi_dBm/10))
+deadline: unsigned long long = Time after which this sample shall be deleted
+next: struct Rssi_sample = next RSSI sample

Figure 1.1: Simplified linked list that each AP uses.

1.1.2 Keep it up-to-date

In the previous section, we quickly presented the timer attribute of an UE measurement. We now go deeper and show why this is important.

Since an UE is moving all the time – walking, going from the door to the window, etc. – keeping its measurements for too long will distort its localization.

Therefore, in order to ensure the best possible accuracy, each measurement has a timer and as soon as it reaches a life-span higher than a second, it is removed from the list.

It is possible to constantly remove the old measurements using a daemon thread that checks the 'deadline' attribute of a list element as shown below:

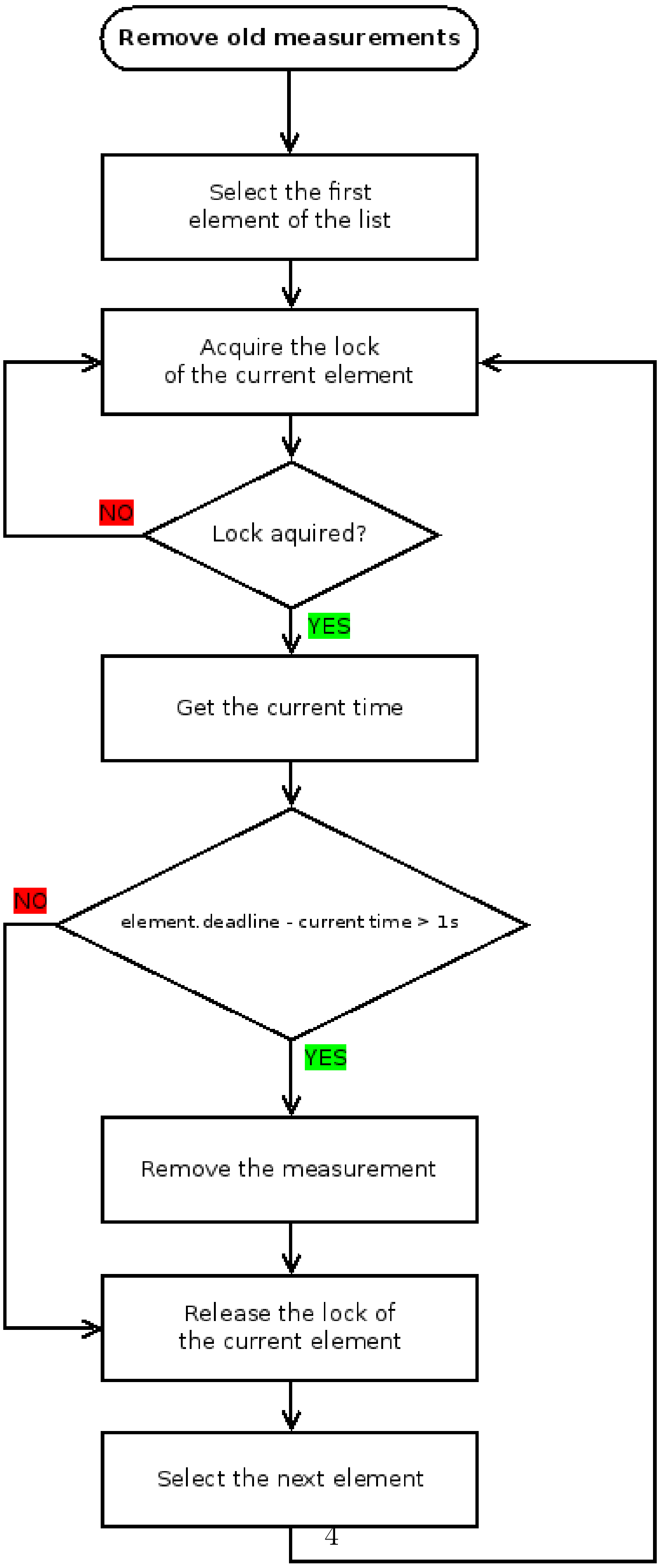

Remove old measurements
Select the first element of the list
Acquire the lock of the current element
NO
Lock aquired?
YES
Get the current time
NO
element.deadline - current time > 1s
YES
Remove the measurement
Release the lock of the current element
Select the next element

Figure 1.2: Remove old measurements in the list.

Due to a lack of time, we were not able to complete the implementation of this feature in our project.

1.1.3 Build the response

In order to prepare the communication between the AP and the Positioning Server, the AP list also gives a function that will convert the list into a JSON formatted response.

Named ‘build_buffer‘, this function looks for a specific MAC address in the list. It then retrieves the RSSI values the AP has and computes their average value.

Then, using the ‘snprintf‘ standard C function, it converts the result into a JSON dictionary, fully prepared to be sent back to the PS.

1.2 Passive Sniffer

We have seen how the AP stores the data. Let us now see how it finds each value.

1.2.1 Listen to Wi-Fi packets

In order to find the MAC addresses and the RSSI values of each device, it needs to sniff the network for any Wi-Fi signals sent by the UEs. The AP achieves this sniffing by using the *PCAP* library.

It sets up a hook on its wireless interface in order to intercept any Wi-Fi packets on the network like shown in the scenario below:

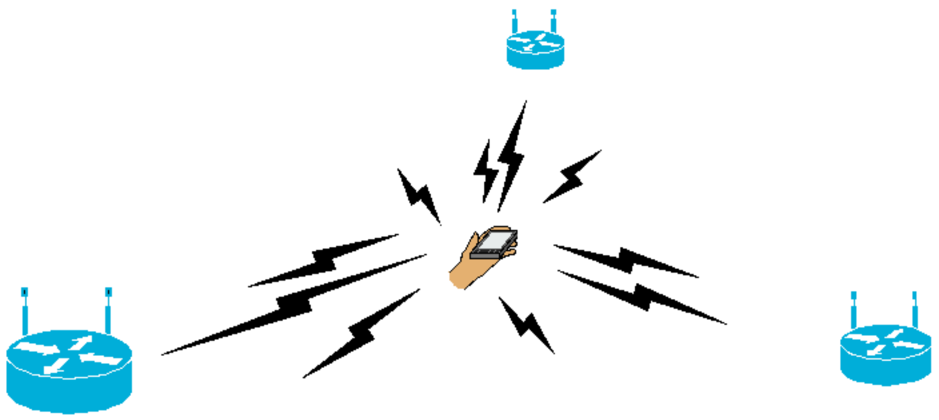


Figure 1.3: Each AP sniffs the Wi-Fi packets of the network.

1.2.2 Extract the values

Then for each packet, it extracts the MAC address and the RSSI value fields and stores them into its list.

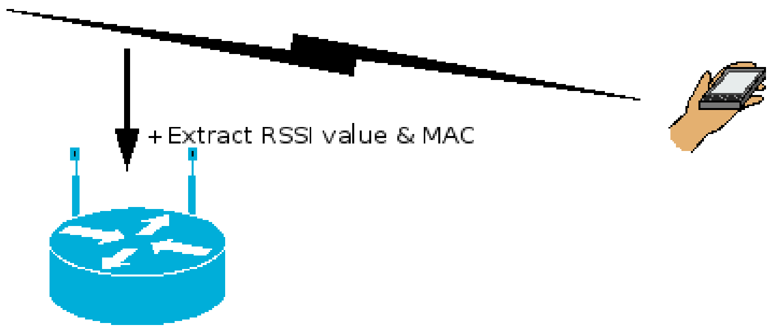


Figure 1.4: For each packet, the AP extracts two fields.

For some times, we had a bug when using the *PCAP* library. The sneakiest problem we have met occurred because of the version the *PCAP* library that was running on the AP.

Indeed, for some versions, a field was added in the header and triggered a buffer overflow when trying to retrieve the different informations we were looking for.

1.3 HTTP Daemon

Last but not least, the AP needs a way to communicate with the PS. This section describes how we manage to set up a communication system between both.

At this point, the AP is able the measure every RSSI values for any devices on the network and stores them. It is also able to convert them to a JSON formatted dictionary.

So in order to send that dictionary to the PS, we have set up a HTTP daemon using the *MicroHTTP* library.

When being fired up, it initializes that daemon using the *start_ microhttpd* function. As a callback function, the AP passes the *connection_ callback* one that reacts to any *?mac=* HTTP requests.

Hence the AP is now able to communicate with the PS. The *connection_ callback* is triggered each time the server asks the measurements of an UE. It then processes the list and builds the response.

1.4 Run the program

We now have every parts of the AP – a data structure, a sniffer and a HTTP daemon – except how to configure and install it on the AP itself. This section answers theses questions.

1.4.1 Configuration

The only configuration our project requires concerns the HTTP daemon and the interface on which to set up the sniffer.

The *http_ daemon.h* file defines two constants:

1. The port on which the daemon will listen to. By default it is set to *8080*
2. The MAC address of the AP.

Then the interface can be set up in the *main.c* file.

1.4.2 Installation

Then, in order to create the program, we provide a *Makefile* along with the source files. Using the command *make all* will generate the binary (it requires the *OpenWRT* toolchain).

Then the binary has to be copied onto the AP (using the *scp* command for instance).

Last, the command *./ap_ daemon.bin* as to be run in order to run the binary on the AP.

Chapter 2

Positioning server

In this chapter, we will speak about the positioning server part of our project.

2.1 Tools and versions

Before seeing the server itself, lets see the different tools used for its creation.

2.1.1 Java

This server is coded in Java 7, the programming language of Oracle. So the different servlets must use TomCat 7.

2.1.2 Simulation scripts

In order to test the server, without having to setup all the elements of the project (Access points, mobile device), three Python scripts have been written:

AccessPoint Simulate the response of an access point.

MobileDevice (Calibration) Simulate the calibration request of a mobile device.

MobileDevice (Localisation) Simulate the localisation request of a mobile device.

2.2 Programm

Now we will look at the server.

2.2.1 DAO

A major element of the server code is the Data Access Object (DAO). It is the link between the database and the code. It allows the programm to manipulate data as it was objects, without having to care about the database. The DAO creates objects from information in the database, and updates information in the database from objects.

The major advantage of this concept is the possibility to change the way to store data easily. Only the DAO interface needs to be modified in case we want, for example, to use a NoSQL database.

2.2.2 Calibration

The Calibration servlet is called by mobile devices to put calibration information in the server database.

How the servlet works:

1. A mobile device send a calibration request, with a map id and coordinates.
2. The servlet get a list of all routers present in database.
3. If there is enough routers, it goes on, else it stops.
4. It sends RSSI requests to router and wait no more than 500 ms for the response.
5. If there is enough useful responses, it stores them in the database (RSSI table), else it stops.
6. It sends a response to the mobile device.

2.2.3 Localisation

The Localisation servlet is called by mobile devices who want their localisation.

How the servlet works:

1. A mobile device send a localisation request, with no parameters.
2. The servlet get a list of all routers present in the database.
3. If there is enough routers, it goes on, else it stops.

4. It sends RSSI requests to router and wait no more than 500 ms for the response.
5. If there is enough useful responses, it stores them in the database (TempRSSI table), else it stops.
6. It gets every location calibrated from the database and calculate the distance with the data in the TempRSSI table, for each location.
7. It returns the closest location to the mobile device (map id, coordinates).

2.2.4 Computation of a location

The two servlets uses the same function for calculating the RSSI distance. This function is used to compute the distance between the new location and all the known locations. During a localisation request, the nearest point is calculated and returned to the mobile device.

2.3 Database

The database of the server is shared between the servlets. Its structure is almost the same as in the subject. There is only one addition: an “ip” column in the AccessPoints table (it is more reliable than using the id for the last part of the ip).

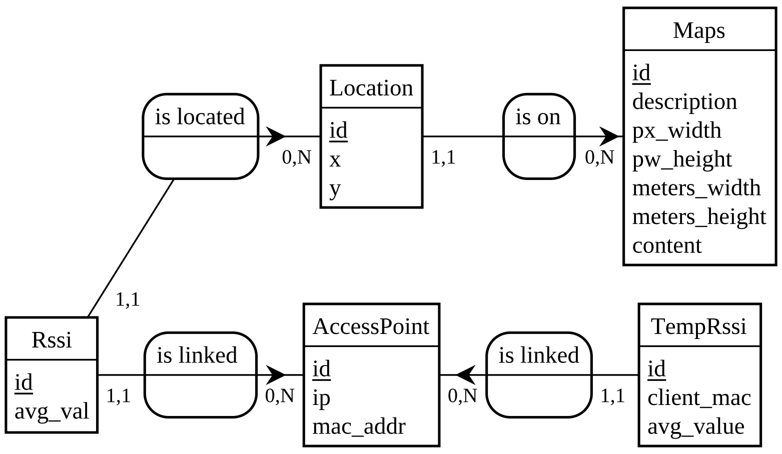


Figure 2.1: Database

2.3.1 Creation script

In order to create the database, a postgresql script has been created. It creates a user “lo53” with a password “lo53” and a database “lo53_rssi”.

2.3.2 Filling

The filing of the database has to be handmade, except for the Maps table. As this table contains binary data, it is quite difficult to do it manually. So we wrote a filling script to do this. This script put maps pictures in the database, and get informations about maps from text files like this one :

```
[picture]
description: This is the description for the H_RDC picture
meters_width: 2.5
meters_heigt: 3.5
```

The size in pixel of the picture is automatically determined, thank to the PIL library. The addition in the database uses the psycopg2 connector.

2.4 Configuration file

Some values of the positioning server may need adjustments. These values should not be hardcoded, because it is not practical at all. So the servlets share a configuration file named *config.properties*. Below is the content of this file.

```
# Database
url = jdbc:postgresql://localhost:5432/lo53_rssi
user = lo53
password = lo53

# Access points port
ap_port = 8080

# Access points timeout (ms)
ap_timeout = 500

# Minimum number of Access Points needed
min_ap = 1

# Minimum number of samples sent by the Access Points needed
```

```
min_samples = 1

# Minimum number of temporary RSSI measurements needed
min_measurements = 1
```

This file already exists, and can be found in the WEB-INF/classes folder of the positioning server. Without it, servlets won't be able to work correctly.

Here is the explanation for each value of this file :

url, user and password These values are used to configure database information.

ap_port It is used for the communication between servlets and access points. We assume that all access points use the same port.

ap_timeout It indicates how long must servlets wait for access points responses during RSSI requests.

min_ap It represents the minimum number of access points needed in order to make a RSSI request. If there is not enough router, no request will be made.

min_samples The minimum number of RSSI samples by access point required.

min_measurements The minimum number of temporary RSSI measurements needed in order to calculate a RSSI distance or validating the RSSI measurements for a location.

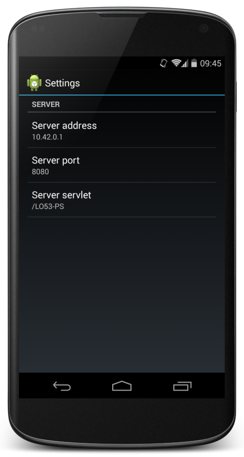
Chapter 3

Android

In this chapter we will talk about the two applications used by the different users to communicate with the server.

3.1 Settings

First, for this two applications we designed a preference activity which allows the user to change different values to access the server. Indeed the address, the port and the servlet to use can be changed.



SERVER

Server address
10.42.0.1

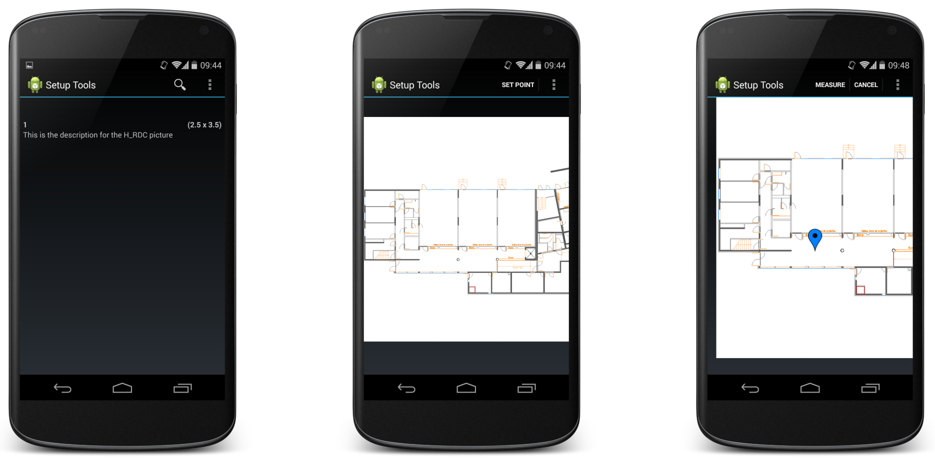
Server port
8080

Server servlet
/LO53-PS

Figure 3.1: Settings activity

3.2 Setup Tool

Now, we will talk about the setup tool application which allows a network administrator for example to set some calibration points in the building. On the next figure you can see the different phase to use this application.



1 (2.5 x 3.5)
This is the description for the H_RDC picture

(a) Choose your map (b) Display the map (c) Set calibration point

Figure 3.2: Setup Tools application

- First, when you launch the application there is a list of all map available on the server which is displayed. It contains the id, a description and the size of the map in meters. Then you just have to select the correct one. It is also possible, when you know the id of the map to use the search bar. Indeed you just have to put the correct id and it will display the map just like if you choose it from the list. (See Figure 3.2a)
- Then when the map is displayed you have to click on the "set" button to be able to add a calibration point. (See Figure 3.2b)
- Once you clicked on the "set" button you can set your calibration point and click on "measure" to send all the information to the server. It's also possible to cancel using the "cancel" button which remove your calibration point (on the map). (See Figure 3.2)

3.3 Positioning

Now we will talk about the positioning application which allow a user to be located in a building according to the calibration points set with the previous application.

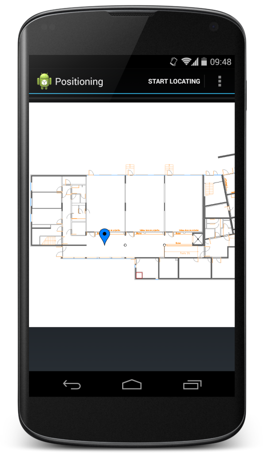


Figure 3.3: Positioning application

This application is very simple. The user just have to click on "Start locating" and the application sends a positioning request to the server and display the result on the screen (map to display, position).

3.4 RSSI values

For this two applications the access points require a maximum of RSSI values. If the mobile does not communicate, the access points do not retrieve RSSI values which prevents the applications to work correctly. So in both application we ping the server but also launch a wifi scan which allow the access points to retrieve about 50 RSSI values.