Project Report

Wi-Fi Based Positioning System

THUILLIER Etienne GI04 LEIM
TISSERAND Geoffrey GI04 LEIM

Sommaire

I) Introduction

The purpose of this project is to built a **full Wi-Fi based positioning system** restricted to a LAN usage.

The system is based on **3 different parts** :

- Wi-Fi Access Points [AP]
- a positioning server
- an Android device

There are **two phases** to consider in this system. The first phase called **offline** (calibration) supposed to build a RSSI map. The second one called **online (positioning)** supposed to locate the device in an indoor environment (room).

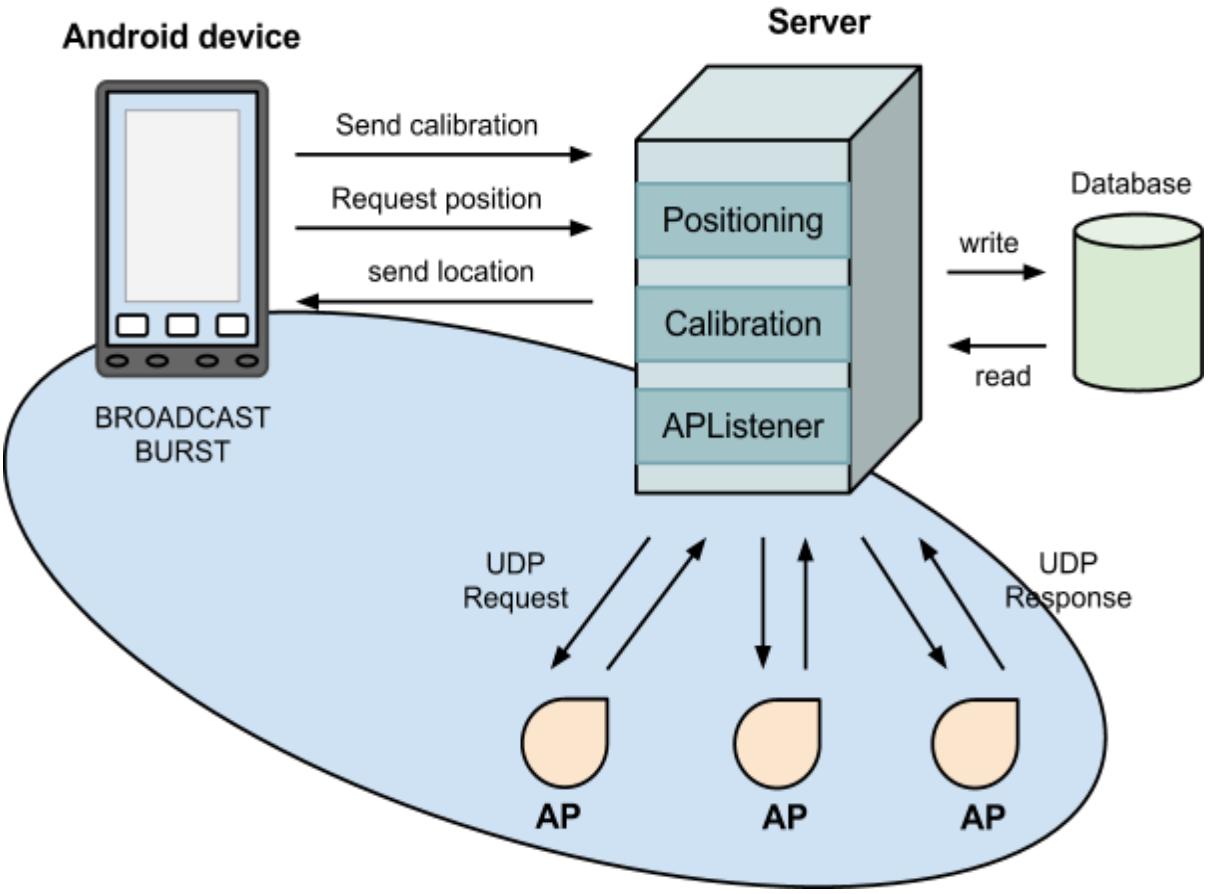
The whole project was built on a **Debian Virtual Machine** launched with **Virtual Box** on a regular Ubuntu machine to allow us to develop with all the environments and the IDE needed.

We used **OpenWRT** for the access point. We simply created an executable able to run on the access point. The C files are compiled in order to be runnable from the OpenWRT access point; the compiler used in this case was given and was a specific gcc mipsel. It was found in this subdirectory: openwrt/staging_dir/toolchain-mipsel_gcc3.4.6/bin/mipsel-linux-gcc

Eclipse IDE was used to create the different **Java Servlet** based on a **Tomcat** server and connected to a **postgresql** database. This database is used to store data coming from the APs and thus enable the positioning process.

We also used the **Android SDK** with the **ADT plugin** for **Eclipse** to create our different android applications.

The following diagram show the whole communication types between each part of our system :



In this report we are going to describe how we built our system, which features are fully functional and which are not, explaining why, where we encountered difficulties and the solutions we chose to achieve this project.

Finally we will do an analysis of our work and conclude on this experience.

II) Features

First here is the list of all asked features with their state and the number of hours spent by x student of the team.

1) Access point

Understand the conception of the project (Embeded virtual machine, linksys AP, what are the steps for calibration and positionement)	functional	2man/360min
Receive data form device using pcap	functional	1man/60min
Extract Rssi information from packet	functional	1man/20min
Extract and parse Mac address	functional	1man/30min
Manage the time convertion problems (timeval)	functional	1man/20min
Store Mac informations	functional	1man/120min
Store Rssi informations	functional	1man/10min
Manage the Rssi informations (add, delete...)	done	1man/10min
Manage the Mac informations	done	1man/10min
Manage UDP packet (calibration or location)	functional	1man/20min
Extract and parse informations from UDP	not implemented	1man/10min
Send UDP packet back to server	functional	1man/10min
Manage the threading between the Devices listening and the UDP socket	functional	1man/20min
Manage mean rssi value for mac adress	done	1man/20min
Manage concurrent access to rssi storage	not implemented	

TOTAL AP	1man/740min

2) Positioning Server

a) Database

Database created	functional	1man/10min
User created	functional	1man/10min

Tables created and accessible	functional	1man/20min
Testing/debugging/rights problems		1man/40min
TOTAL		1man/80min

b) Calibration servlet

Can be reached by android app / or anything else, using get request	functional	1man/20min
Pass and Retrieve coordinates sent by android app in GET parameters	functional	1man/20min
Retrieve the mac address from /proc/net/arp table with the request IP	functional	1man/40min
Send a GET;X;Y;mid;ZZ:ZZ:ZZ:ZZ:ZZ:ZZ message to all the APs	functional	1man/20min
Send back the coordinates received to notify android app that it worked	functional	1man/20min
Testing/Debugging/Adding log info		1man/40min
TOTAL		1man/160min

c) Positioning servlet

Can be reached by android app / or anything else, using get request	functional	1man/10min

Retrieve the mac address from /proc/net/arp table with the request IP	functional	1man/20min
Send GET;XX:XX:XX:XX:XX:XX message to each APs	functional	1man/10min
Sleep during 500ms	functional	1man/5min
Connection to database to look for temps rssi values	functional	1man/20min
Algorithm to determine location from rssi values	functional	1man/40min
Send location back to the android device (random if not found)	functional	1man/40min
Testing/Debugging/Adding log info		1man/30min
TOTAL		1man/175min

d) APListener

Running in background while(true)	functional	1man/20min
Get UDP packets from APs	functional	1man/30min
Find if it's a Calibration, a Positioning or a SPAM response by counting the size of the splitted tab	functional	1man/10min
Extract values from packets	functional	1man/10min
Connection to database	functional	1man/20min
Store information into database	functional	1man/40min
Testing/Debugging/Adding log info		1man/30min
TOTAL		1man/160min

TOTAL SERVER	1man/575min

3) Android device

a) SetUp Tool

Building user interface / preferences	functional	1man/60min
Chose between different maps	not implemented	0man/0min
Turn on/off active mode + cancel button	functional	1man/10min
Set and display a chosen location on the map by touching the screen	functional	1man/20min
Calculate all coordinates (point selected in pixel, location on the room in cm)	functional	1man/10min
Burst packets to APs	functional	1man/20min
Send location to the server (measure button)	functional	1man/20min
Ensure location are correctly sent by looking into server response	functional	1man/10min
Testing/Debugging/Adding toast messages		1man/30min
TOTAL		1man/180min

b) Positioning

Building user interface / preferences	functional	1man/30min
Burst packet to APs when start locating button hit	functional	1man/5min
Sending positioning request to the server	functional	1man/10min
Looking server response	functional	1man/5min
Set the map pin to the location sent by the server	functional	1man/10min
Testing/Debugging/Adding toast messages		1man/20min
TOTAL		1man/80min

c) Joining both apps

Joining apps	functional	1man/40min

TOTAL		1man/40min

TOTAL ANDROID APPS	1man/300min

Testing everything together / Debugging	2man/600min

TOTAL PROJECT TIME	2man/2215min

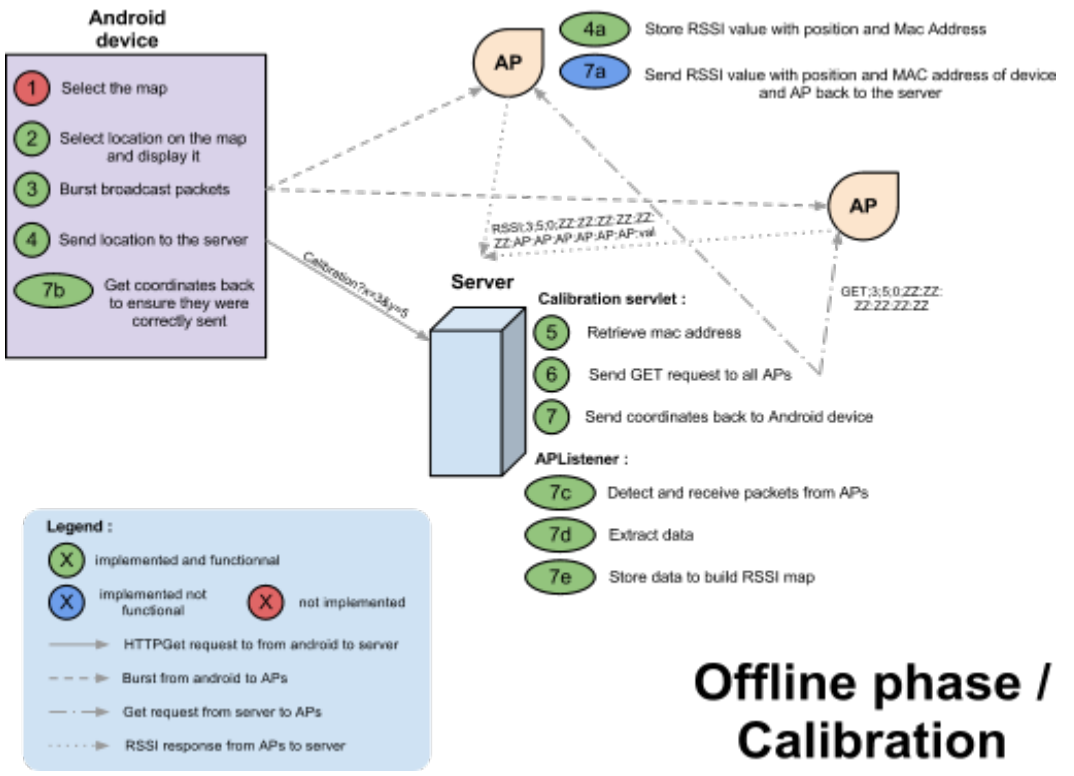
4) Global features diagram

In order to understand where we have communication problems please find the following graphs where you can see a summary of our features.

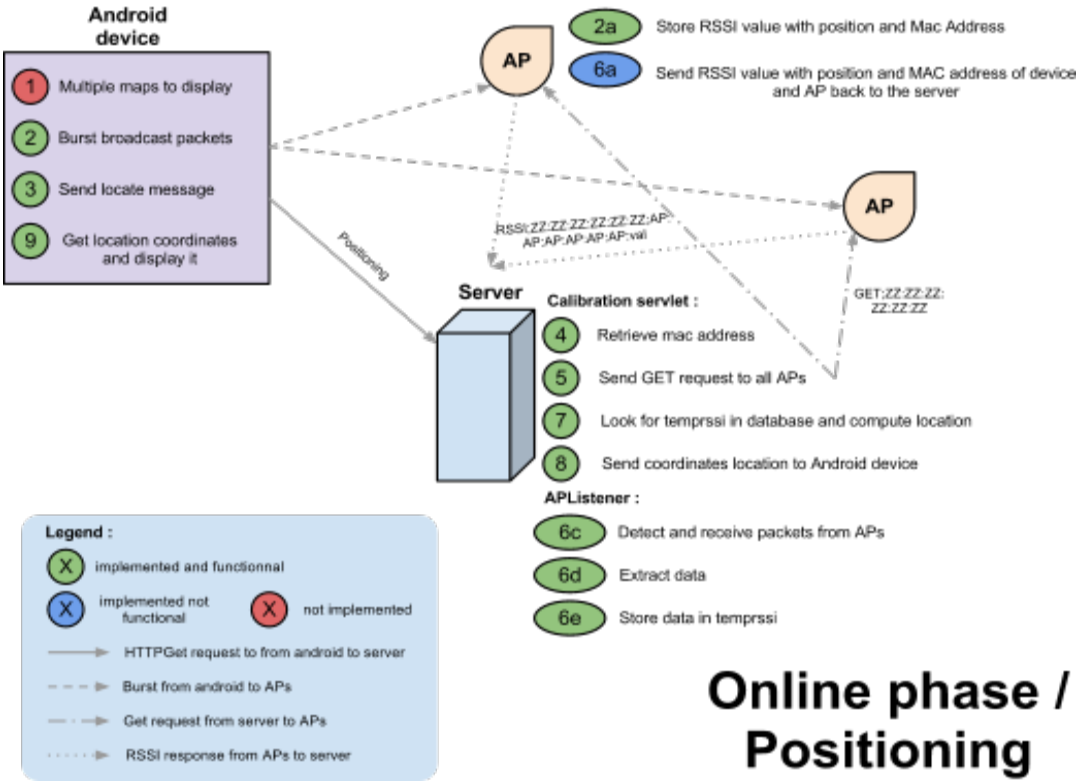
We split the diagram into two images for the two phases in order to make it as readable as possible.

You can follow the algorithm with the number placed before the instructions.

When the number is completed by a letter it's because we suggest that the tasks are processing in parallel.



Offline phase / Calibration



Online phase / Positioning

III) User manual

To install every part of our system you'll need a machine where **Eclipse IDE**, a **Tomcat** server and a **postgresql** database are installed and ready to use. You'll also need some **Linksys routers**.

This user guide will help and explain you how to use our system.

1) Deploying the APs

Note : Sources are located in /**AP/** folder in the root of our zip file.

You have to deploy this software to every routers you need. The AP needs to be plugged to your machine.

To send the C program to the AP, you first need to install the virtual machine (Debian) to be able to compile the files for the linksys AP.
The compilation is made with a makefile. The compilation line is:

```
$(GCC) $(ALLFLAGS) AP_algorithm.c -o AP_executable
```

with:

```
PREFIX=/home/lo53/openwrt/staging_dir/toolchain-mipsel_gcc3.4.6
GCC=$(PREFIX)/bin/mipsel-linux-gcc
LD=$(PREFIX)/bin/mipsel-linux-gcc
ALLFLAGS=-Wall -Werror -O2 -I$(PREFIX)/include/ -I$(PREFIX)/../mipsel/usr/include/ -L$(PREFIX)/lib/ -
L$(PREFIX)/../mipsel/usr/lib/ -lpcap -lm -pthread
```

Then, you have to send the file into the AP by using a SSH request :

```
scp ./AP_executable root@192.168.1.1:AP_executable
```

Where the first AP_executable is the name of the compiled C file, and the second one is the name in the access point of the executable. You will need the password for the AP; generally it's 'admin'

Finally if you want to test it you will have to connect to the AP and manually and start the software : that means use SSH link. To do this you have to type the following command :

```
ssh root@192.168.1.1
```

Then, once you are in the AP you should see the following

```
root@LinkX:~#
```

You need to set the AP ready for and executable; just type:

```
wlc monitor 1
```

And then you can launch the executable by typing:

```
./AP_executable
```

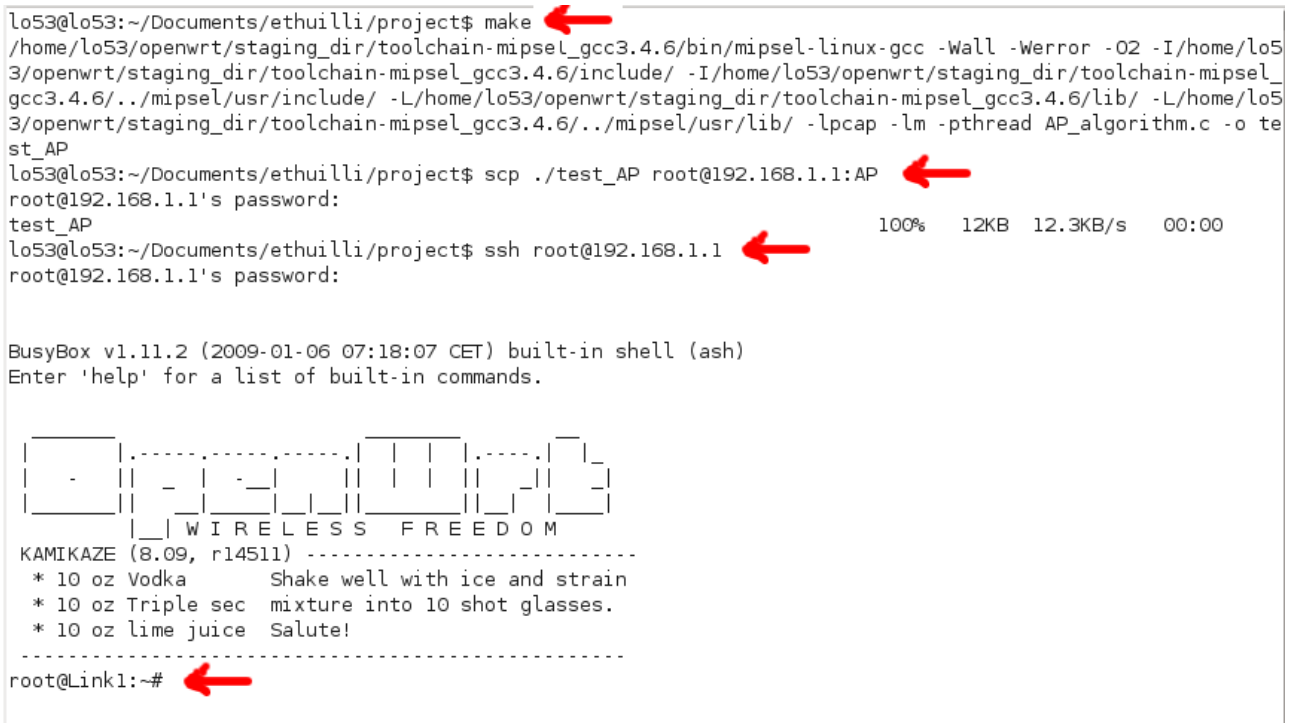
To simplify all these tasks we created a bash file to compile send the file to the AP and connect to the AP.
You just need to type this in the command line

```
./compile AP_executable
```

bash file:

```
#!/bin/bash
if $all;
then
   make   #compile
   scp ./"$1" root@192.168.1.1:$1   #send file to the AP
   ssh -t root@192.168.1.1 #connect to router
    rm $1
fi
```

The total process should look like this:



2) Deploying the servlets

Noe : For the database, you just need to copy/paste our **CREATEDABASE.sql** script in the root of our zip file into a **postgresql** database called lo53 owned by lo53 user with password lo53 (or change parameters in the APListener as explained later in this section).

Note 2 : Sources are located in /**LO53-Server/** folder in the root of our zip file.

To deploy the servlets you need to run Eclipse IDE and a Tomcat server configured to be runnable in Eclipse.

Next you just have to **import** the project **LO53-Server**. It is possible that you have error (exclamation point in the project title) because we developed in our own machine. You just have to go into project **properties** > **Java Build Path**. In the **Libraries tab** click on **JRE System Library** and click **Edit** then chose the right alternate JRE.

Once you imported the project, open the **Servers View** into Eclipse and **right click** on the **Tomcat** server then click on "**Add and Remove**" option. In the available list of resources click on **LO53-Server** and Add it to the table in the right by clicking "**Add**" button.

Then you have to **set up** some **parameters** in the **APListener**. These parameters are in the **attributes field** of the **java class** and are used for **database connection** / **sending message**

to APs / Listening APs responses.

Once you have done that, you just have to **build** the three java classes placed in the default package of the LO53-Server project.

Once they are built, simply **launch** the **Tomcat Server** - if it's not already launched by clicking the **launch icon** on the **Servers View.**

Server is now ready to handle android apps requests, to listen AP responses etc...

3) Deploying android app

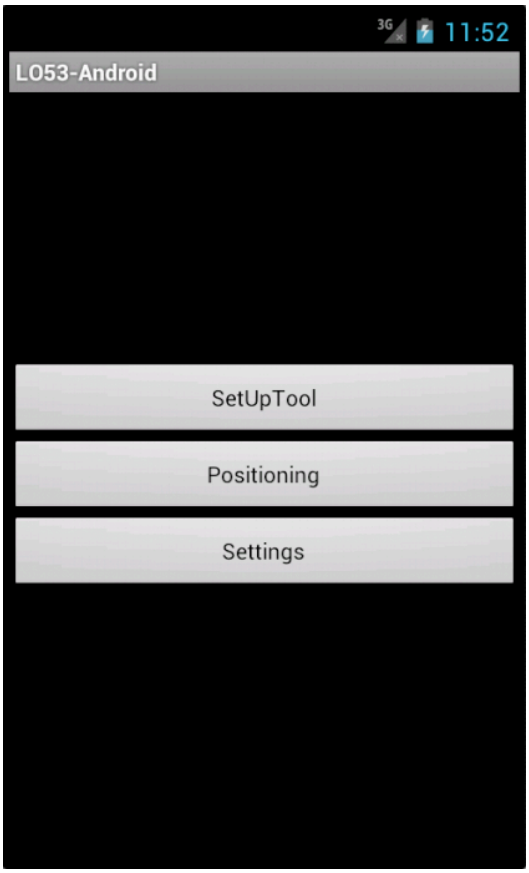
Note : Sources are located in /**LO53-Android/** folder in the root of our zip file.

To install the android applications you have to import the project **LO53-Android** into Eclipse.

The you have to **set up** some **parameters** in the **Main class**. These information are in the **attributes section** of the class and are used to burst packets to APs.

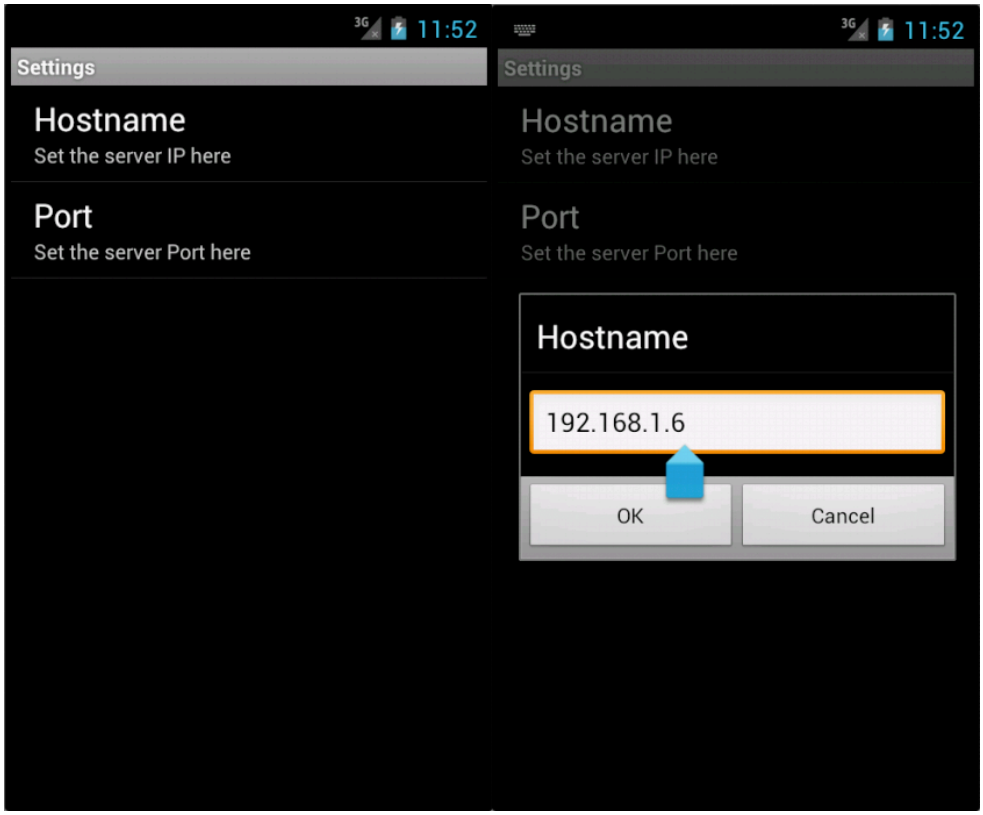
Once you have imported this project and setup these attributes, simply build and launch **Main** classe located in the default package of the android project.

If you have an android device simply plug it into your computer and the application will be uploaded and launched on your device.



a) Settings

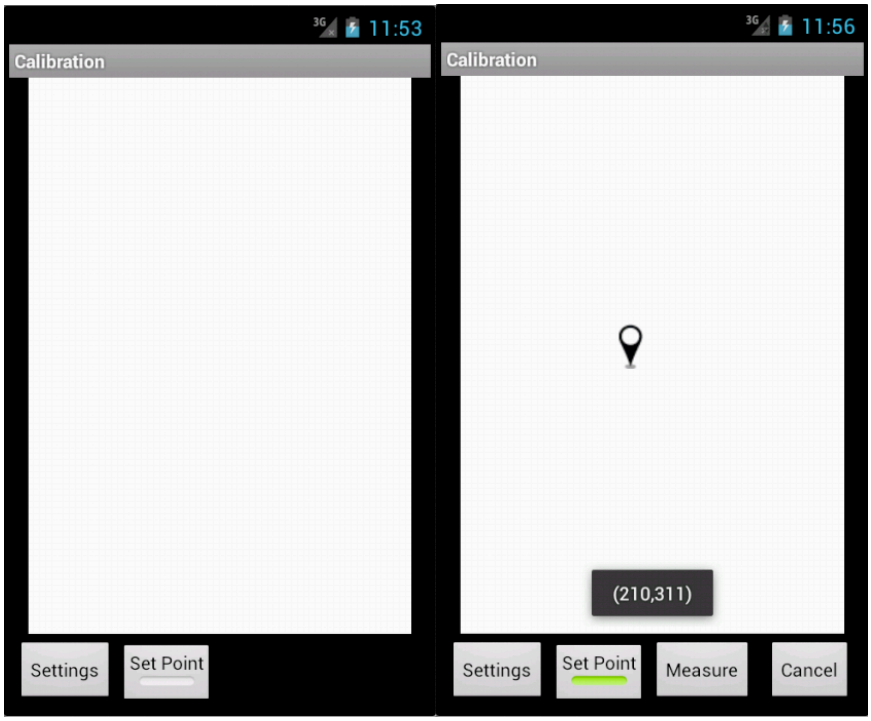
When you launch the application, you can chose to calibrate, to locate or to modify the settings. Going to settings will allow you to **setup server information** (**hostname** and **port** which are persistent thanks to *PreferenceActivity* Android class). **Important**, you have to set these settings the first time you launch the application in order to know which server you want to request (the one with our servlets on it).



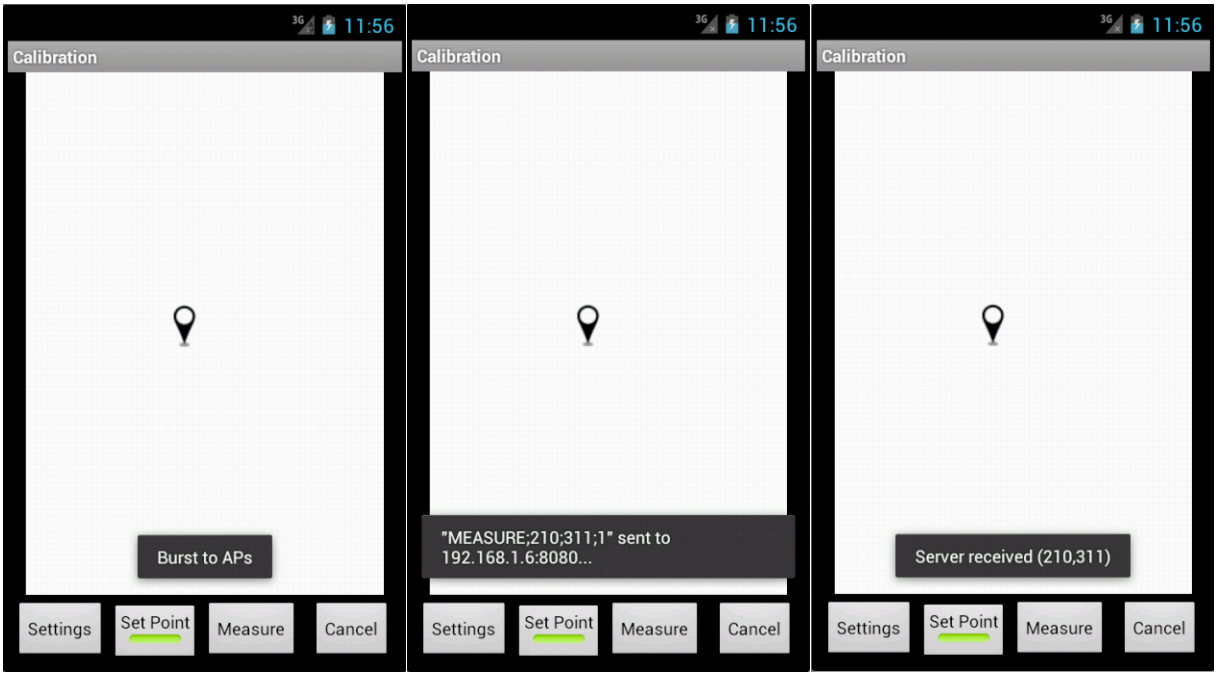
b) Calibration

This part of the application allow an administrator to **calibrate** the APs with the Android device. This tool is supposed to let the user chose between different map scales, but we chose to display only one map by default. When this application is launched, only two buttons are displayed. The first one allow the user to go directly to settings and the second one is a toggle switch which allow the user to enter in the *active* mode.

In the active mode, the user can touch anywhere in the map to set is position on the room. We used *Toast* messages to improve the user interface to show the exact touched location. Two new buttons appears, *measure* and *cancel.*



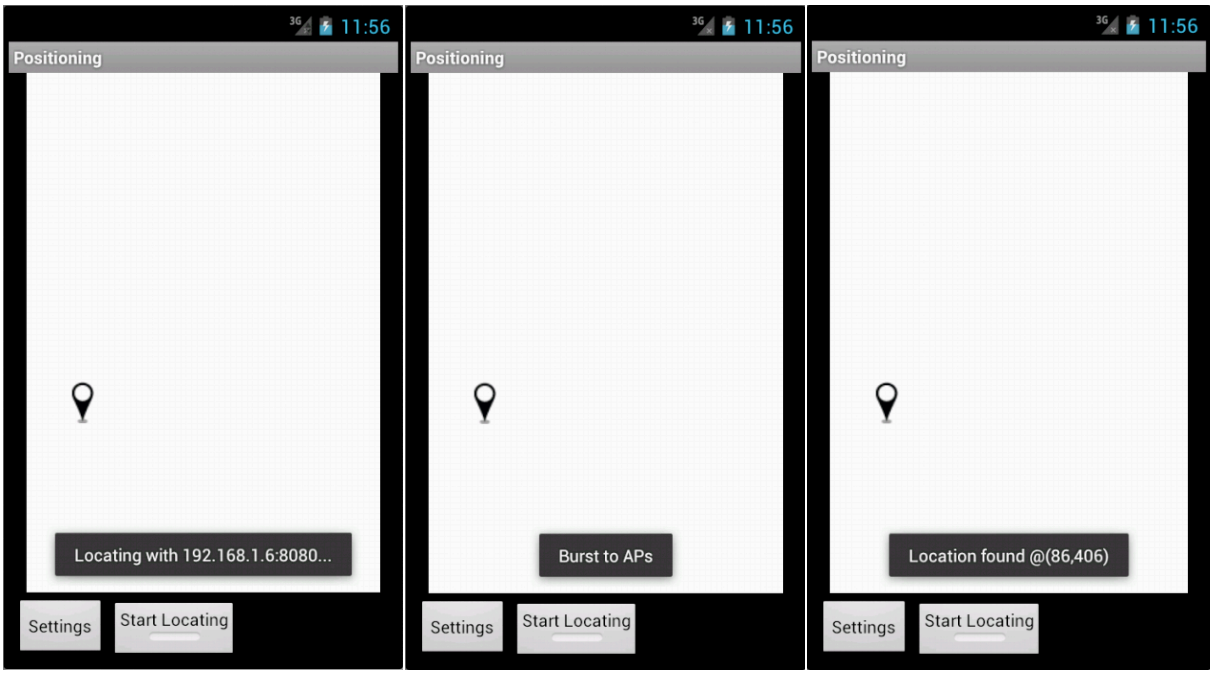
The cancel button turn off the online mode and the measure button send a message to our *Calibration* servlet in order to build the RSSI map. When the server receive the location, it send it back to the android device to be sure the server received it.



b) Positioning

The third part of the application is the positioning tool which display a default map with only two buttons.

The first one, settings works like the first app, and the second one, start locating, is a toggle switch which send message to our Positioning servlet to find the user location in the room. Once the location is sent by the server, a pin is displayed on the map according to the locating algorithm which uses the nearest point in signal strength. If we were unable to locate the device we send a random location to the android device.



IV) Test

We tested our system with only one AP deployed. This AP was plugged in our computer which was running our Tomcat server. We used a laptop with the Android Emulator connected in Wi-Fi to the AP plugged to our server.

First we run our Tomcat server. The APListener was also launched because it implements the *contextInitialized*() method from the *ServletContextListener* class which is called when the server is launched. This method simply start our listening thread (the class itself because we extends from *Thread* class) which calls the run method where our is our code. The server was now ready to handle requests.

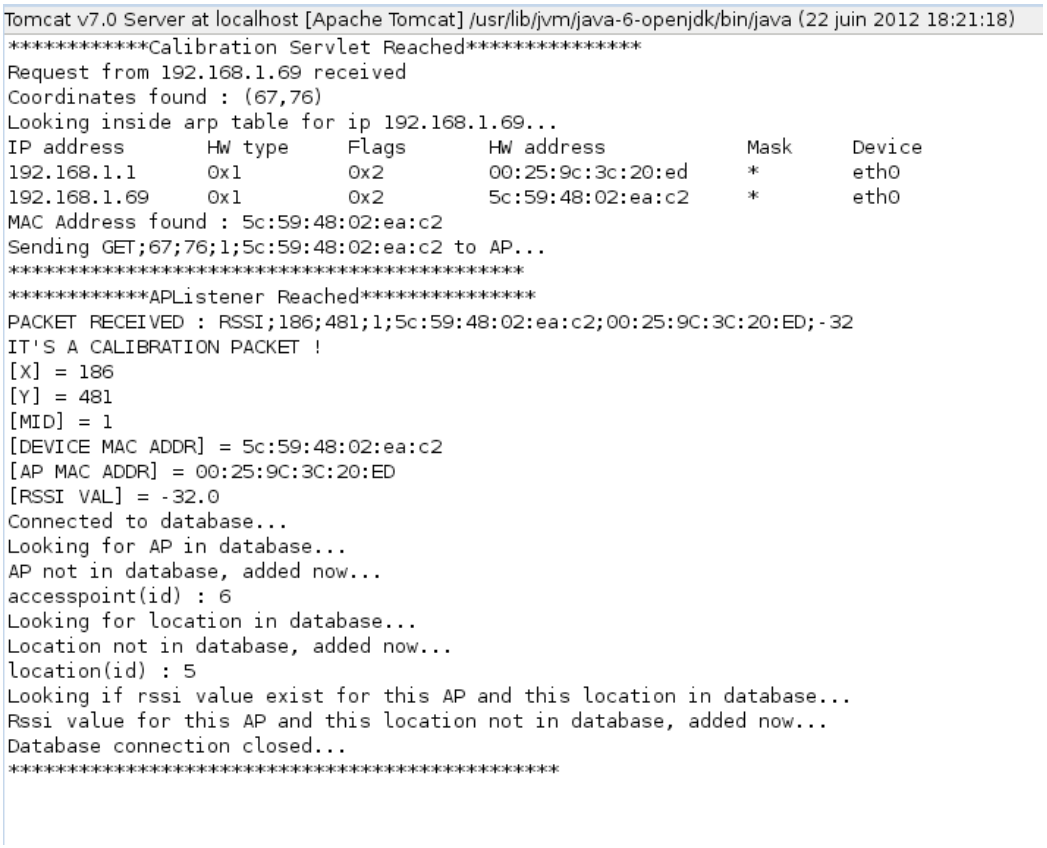
Note : we used 7777 port to send udp packets to the AP. 9999 port to receive packets from AP. And the only one AP had the 192.168.1.1 IP. These parameters are in the APListener class.

Next we launched the AP software. The AP was now listening every data passing on the network looking the mac address and the rssi values.

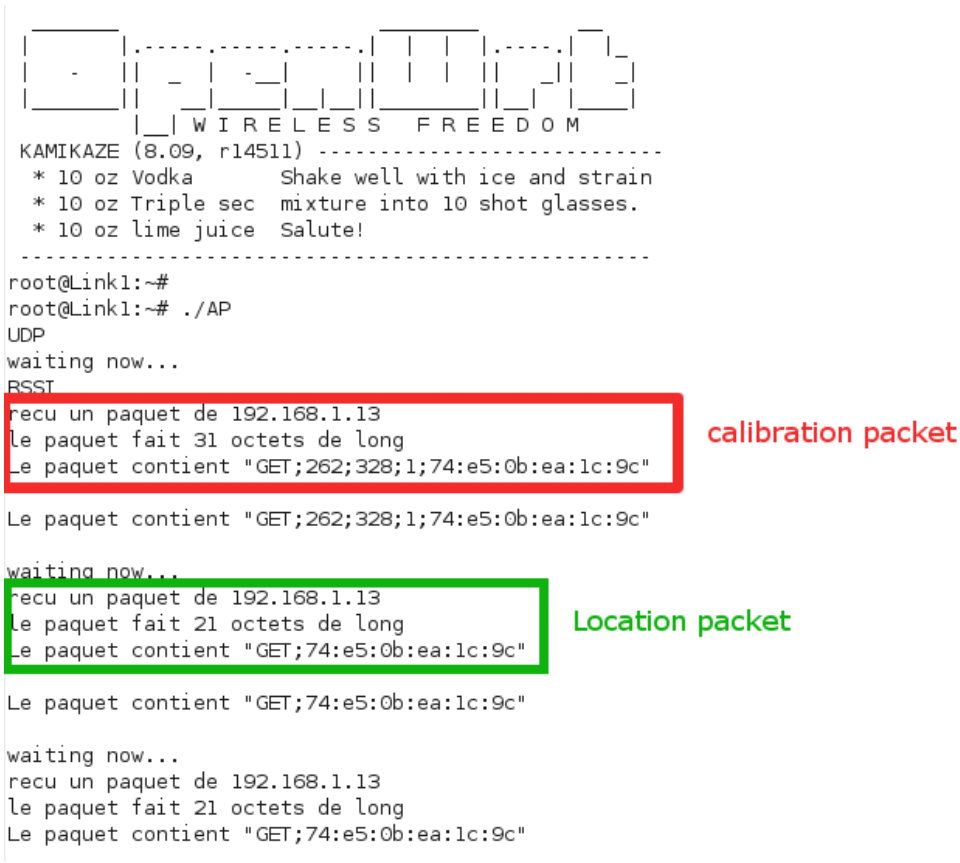
```
Grabbed packet of length 269
Recieved at ..... Fri Feb  6 09:03:54 2009
Mac adress : 74:E5:0B:EA:1C:9C
Dev name : 77:6C:30:00:00:01
Value : -28  dB


Grabbed packet of length 210
Recieved at ..... Fri Feb  6 09:04:03 2009
Mac adress : 74:E5:0B:EA:1C:9C
Dev name : 77:6C:30:00:00:01
Value : -28  dB


Grabbed packet of length 242
Recieved at ..... Fri Feb  6 09:04:03 2009
Mac adress : 74:E5:0B:EA:1C:9C
Dev name : 77:6C:30:00:00:01
Value : -28  dB


Grabbed packet of length 234
Recieved at ..... Fri Feb  6 09:04:03 2009
Mac adress : 74:E5:0B:EA:1C:9C
Dev name : 77:6C:30:00:00:01
re u un paquet de 192.168.1.13
le paquet fait 31 octets de long
Le paquet contient "GET;217;385;1;74:e5:0b:ea:1c:9c"
Value : -28  dB
```

Finally we launched our Android application. We set up the server hostname/port. We started with the SetUpTool. When we had chose our location on the map we sent the measure request to the server.

Note : we used 7331 port to broadcast spam on the 255.255.255.255 IP broadcast address during 10ms. These parameters are in the Main class. We also used 192.168.1.13 which was the hostname of our server on the 8080 port. These parameters are in the android app settings.



The right servlet, the calibration one, is reached. It sends a request to our AP.



The AP detect the packet sent by the Server and it respond using the IP and the port of the request. Then the APListener retrieve response from the only one AP in our system. This response is detected as a calibration response and data is inserted in the database.

```
lo53=> select * from accesspoint;
 id |     mac_addr
----+-------------------
  6 | 00:25:9C:3C:20:ED
(1 ligne)

lo53=> select * from location;
 id |  x  |  y  | map_id
----+-----+-----+--------
  5 | 186 | 481 |      1
(1 ligne)

lo53=> select * from rssi;
 id_loc | id_ap | avg_val | std_dev
--------+-------+---------+---------
      5 |     6 |     -32 |       1
(1 ligne)
```

This screenshot above show that the database is completed with information sent by the AP. *You'll understand why the x and y inserted and received by the listener are not the same that*

the x and y sent by the android device at the end of the Test chapter.

The android app is acknowledged by receiving the coordinates that the server received.

Next we use the positioning tool. By simply clicking on the start locating button.

```
************Positioning Servlet Reached***************
Request from 192.168.1.69 received
Looking inside arp table for ip 192.168.1.69...
IP address       HW type     Flags       HW address            Mask     Device
192.168.1.1      0x1         0x2         00:25:9c:3c:20:ed     *        eth0
192.168.1.69     0x1         0x2         5c:59:48:02:ea:c2     *        eth0
MAC Address found : 5c:59:48:02:ea:c2
Connected to database...
Cleaning temprssi table...
Tempsrssi table clean...
Database connection closed...
Sending GET;5c:59:48:02:ea:c2 to AP...
************APListener Reached***************
PACKET RECEIVED : RSSI;5C:59:48:02:EA:C2;00:25:9C:3C:20:ED;-32
IT'S A POSITIONING PACKET !
[DEVICE MAC ADDR] = 5C:59:48:02:EA:C2
[AP MAC ADDR] = 00:25:9C:3C:20:ED
[RSSI VAL] = -32
Connected to database...
Looking for AP in database...
AP already in database...
accesspoint(id) : 6
Inserting temprssi in database...
Database connection closed...
************************************************
Connected to database...
Retrieve new temprssi...
RSSI found !
APID : 6
CLIENTMAC : 5C:59:48:02:EA:C2
AVGVAL : -32.0
Retrieve nearest loc...
Nearest loc retrieved !
location found @(186,481)
Database connection closed...
*********************************************
```

Again according to the screenshot above, the right servlet is called, requesting the AP. The AP response is catch by our listener. It detect that this is a positioning response and fill the temprssi table.

```
lo53=> select * from accesspoint;
 id |     mac_addr
----+------------------
  6 | 00:25:9C:3C:20:ED
(1 ligne)

lo53=> select * from location;
 id |  x  |  y  | map_id
----+-----+-----+--------
  5 | 186 | 481 |      1
(1 ligne)

lo53=> select * from rssi;
 id_loc | id_ap | avg_val | std_dev
--------+-------+---------+---------
      5 |     6 |     -32 |       1
(1 ligne)

lo53=> select * from temprssi;
 ap_id |     mac_addr      | avg_val
-------+-------------------+---------
     6 | 5c:59:48:02:ea:c2 |     -32
(1 ligne)
```

As you can see it adds a line in the temprssi table. Thanks to this data it can now compute the location on the device. In this case it is very simple, the nearest calibration point of our temprssi is our only one calibration point that we retrieved just before with the setuptool. The servlet found the location @ 186,481, so it sends back to the android emulator the coordinates of this location.

This demonstrate that every part of our system is functional and communicate well. The only thing that is not completely done is that the AP software currently send ‘false’ information because we did not have time to split request and compute mean rssi values. (it is not as simple as in java to split Strings in C). For this example we built ‘false’ responses in the AP software (line 380) with the mac address of our device and false (but coherent) rssi value.

V) Analysis

One of the biggest difficulty was first to understand the global view of the project. The pcap library was not the easiest to install and to parse. One point also was the dynamic C coding which was really off-putting.

We lost some time using PG Admin III instead of standard scripts in command lines. Maybe we could have saved time directly using the terminal to create our database because with PG Admin III all tables were inaccessible from JDBC.

Of course with some overtime we could have implemented the right AP response and the choice between different map scales. If we had more time I think we would have put both servlets into only one servlet which can be called with GET parameters or not to distinguish both online and offline phase. (When called with x and y parameters its a calibration request, when no parameters its a positioning request).

VI) Conclusion

This kind of project is clearly an engineer project: to setup a wifi based positioning system indoor.
Even if we did not managed to make everything by ourselves (Concept, steps, virtual machine, setup of the AP internally) this project required us to understand each step, and finally create this positioning system.

This project was very interesting because of it’s complexity to make everything works together.
We had some problems to make everything works but we were happy when all the parts were ready to communicate.

With this project we were allowed to see each part of a basic full Wi-fi based positioning system in details and to handle and monitoring every communications.
This project was also interesting because we used very different technology for each part of the system improving our skills in these technologies.

It was also a good team project because of it’s ability to be split into parts for each member of the team. It allowed us to work in parallel and compare our results, trying to merge all we have done in the separated parts.