LO53 – PROJECT
INDOOR POSITIONING SYSTEM

Project Members

DUONG Tony
MBOUGUEM Yvon
WABO Joel
ZEUFACK Arnel

Under the supervision of **M. Frederic Lassabe**

Introduction

The goal of this project is to build an indoor positioning system, based on Wi-Fi technology. This project is divided into 3 parts:

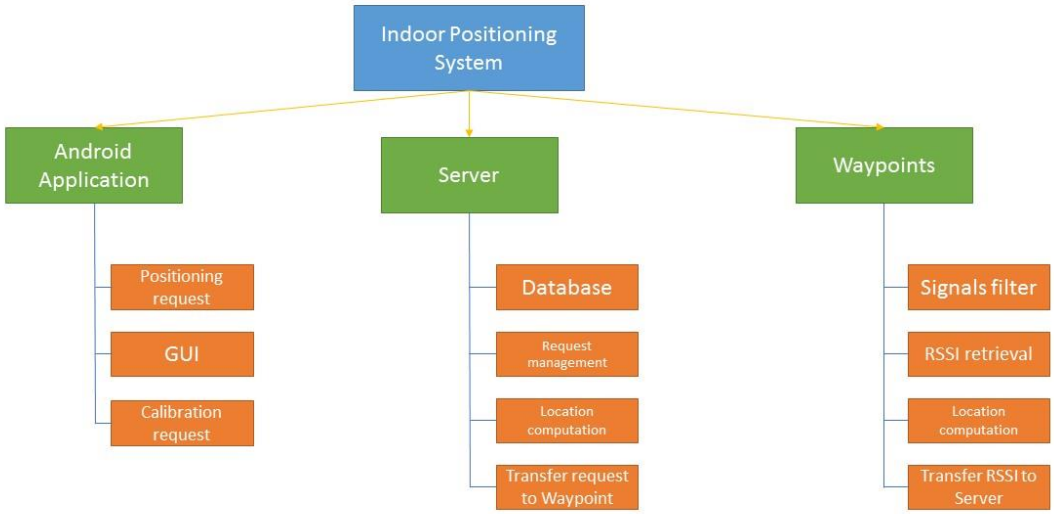
- Wi-Fi access points or waypoints
- Android application
- Positioning server

This report describes our work on this subject. We will explain how we manage to achieve the fixed goals, and the difficulties we have met.

Contents

A. Project management

I. Work breakdown structure



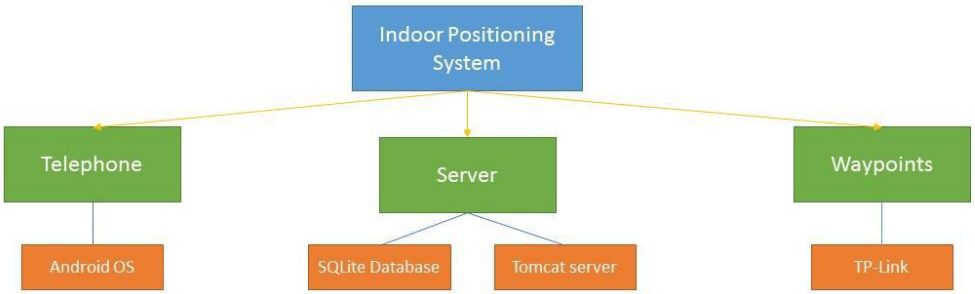
Our indoor positioning system is divided into three parts: Android application, the positioning server and the waypoints.

For the Android application, we have to implement the sending of both the positioning request and the calibration request. On one side, the positioning request is sent to the server by the Android application asking the server to send him back its location. On the other side, the calibration request is used for calibrating the positioning system. Of course, these two functionalities must be implemented along the Graphical User Interface allowing the user to use them.

Secondly, the server is the part responsible for computing the location from the data received from the waypoints (RSSI values and MAC address). So a location computation has to be implemented. But in order to find the location of a mobile, the server has to communicate with a database which contains all the calibration data. Finally, that goes without saying that it also manages the communication between it and the Android application as well as the waypoints.

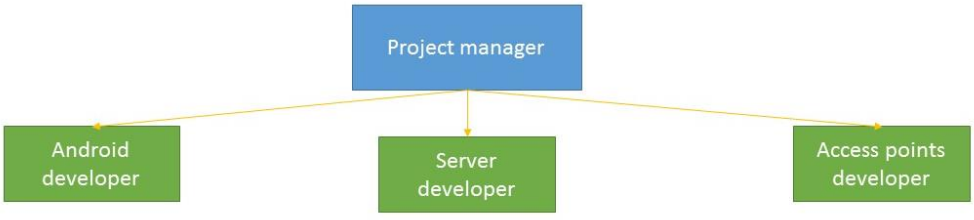
The last part concerns the waypoints. Their role is to retrieve the RSSI values from the Android application and send them back to the server. Then the server will compute the current location using these data.

II. Product Breakdown Structure



As for the equipments, we will use the Android phone and the TP-Link access points provided by the professor. For the server part, it will be run on our own computer.

III. Organization Breakdown Structure



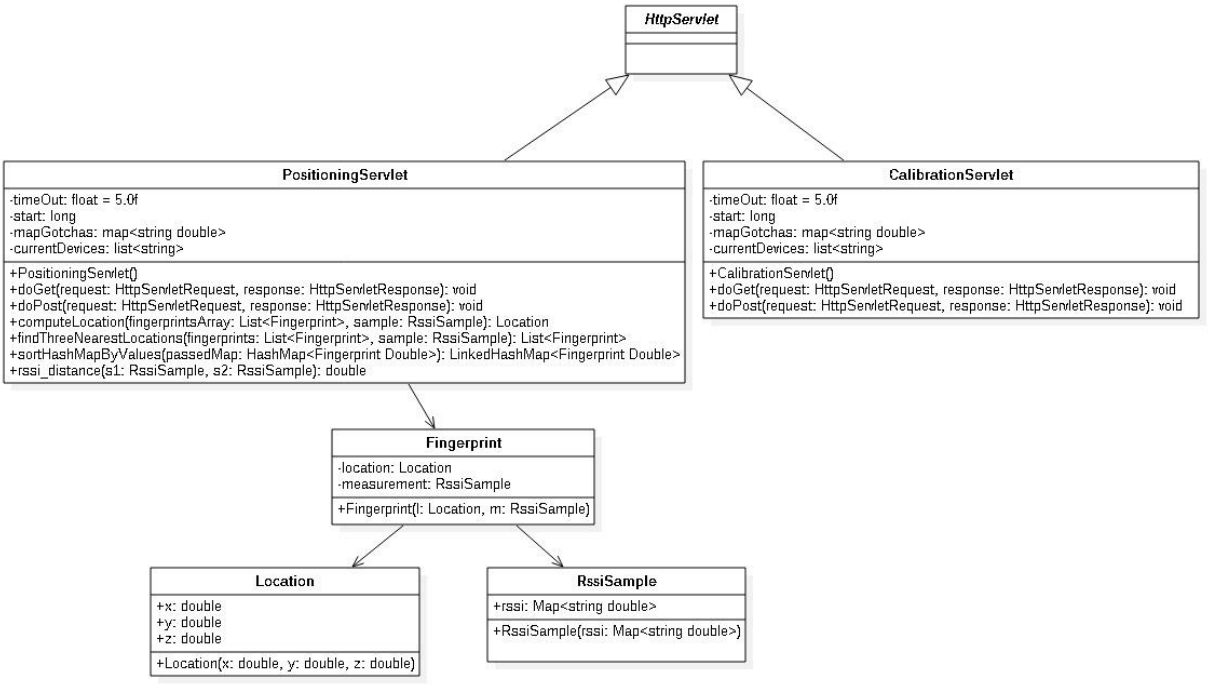
We have chosen to divide the work into three groups:

- Joel Wabo for the Android application development
- Arnel Zeufack for the Access points development
- Tony Duong and Yvon Mbouguem for the server development

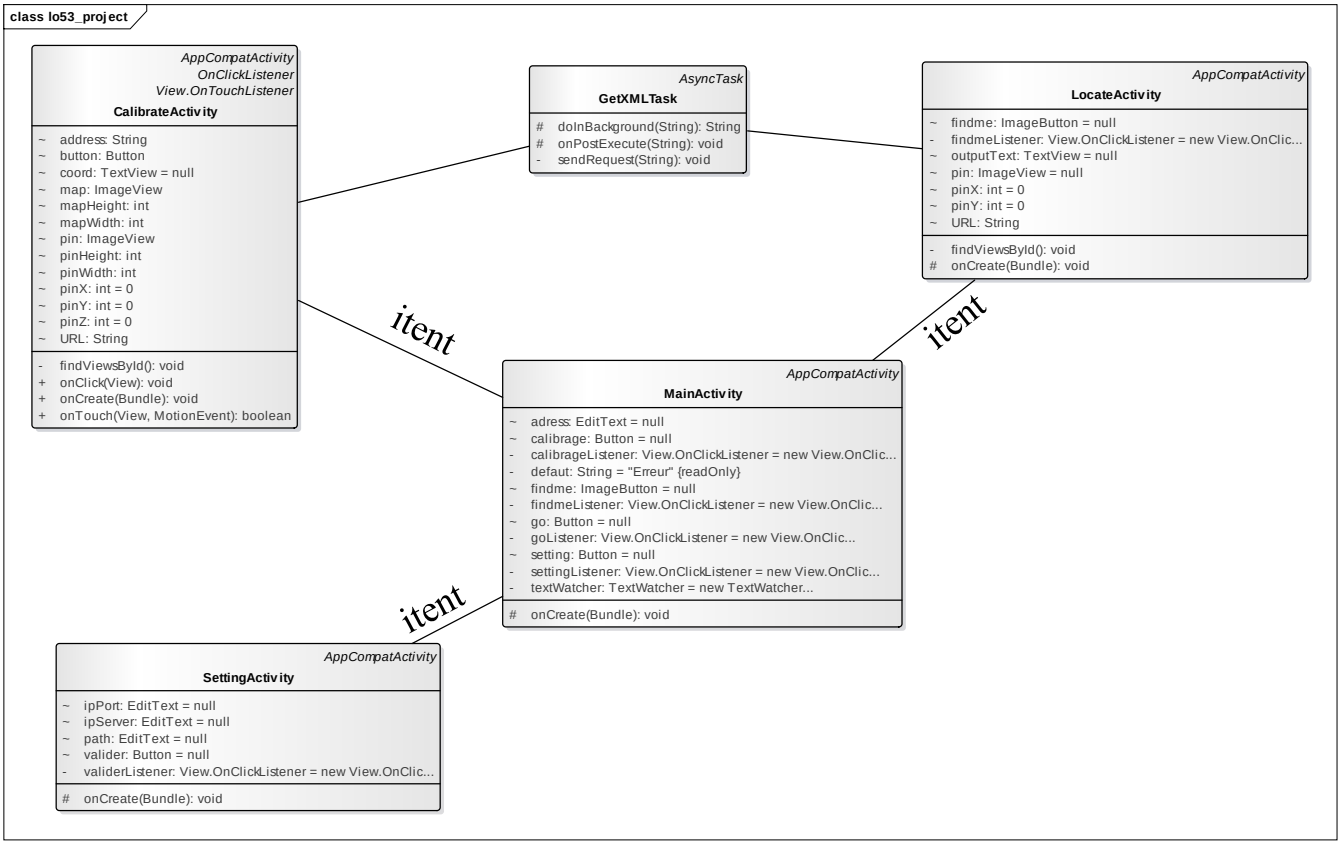
B. Design of the project

1. Class Diagram

Servlet



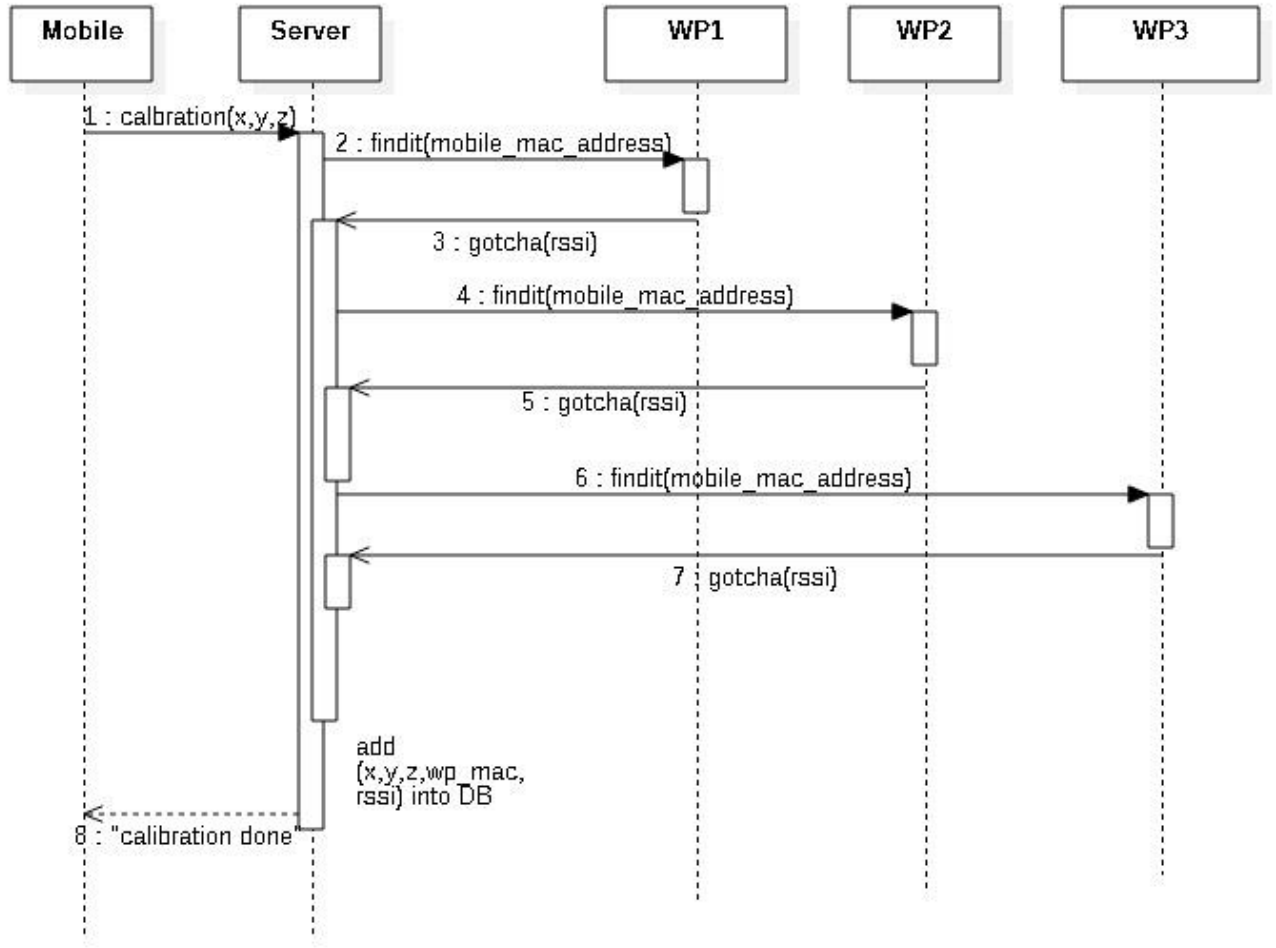
Android application



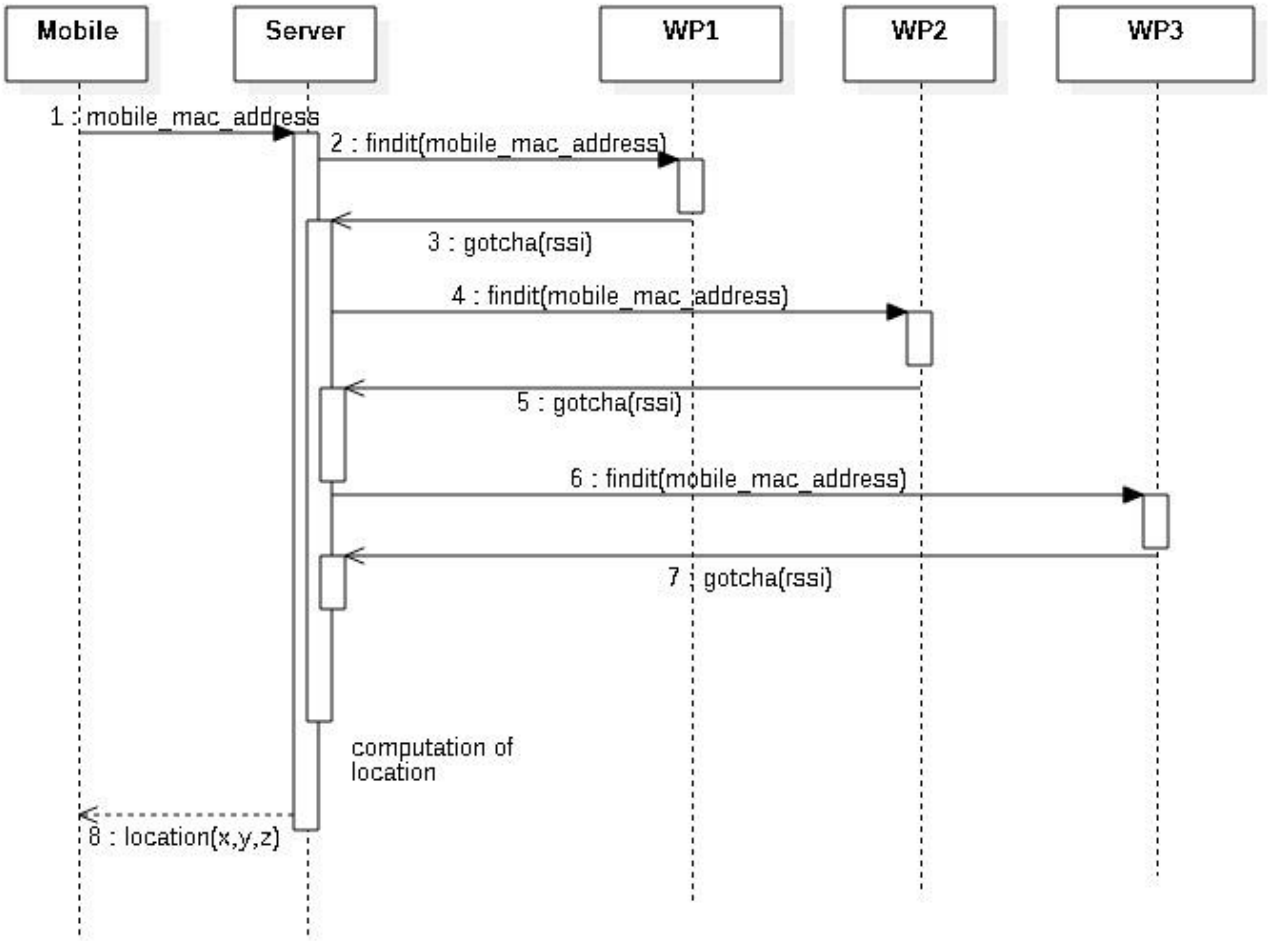
The class diagram is composed of 4 main classes: MainActivity, CalibrationActivity, LocateActivity and SettingActivity. CalibrationActivity and LocateActivity Class extended GetXMLTask class. The android GetXMLTask class helps to run in the background the HTTP request sent to the server so the app will not be blocked on the request.

2. Sequence diagrams

Calibration

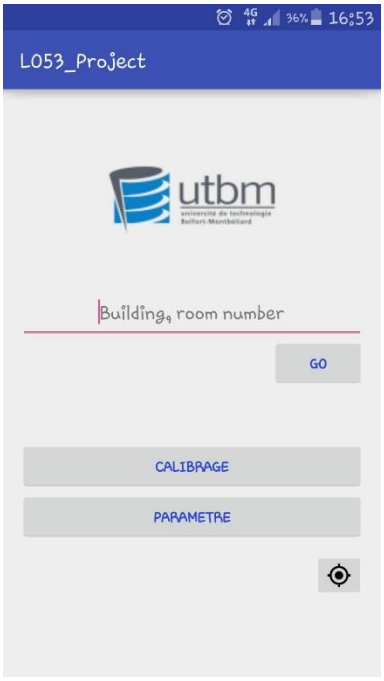


Positioning



C. Indoor Positioning System

I. Android application



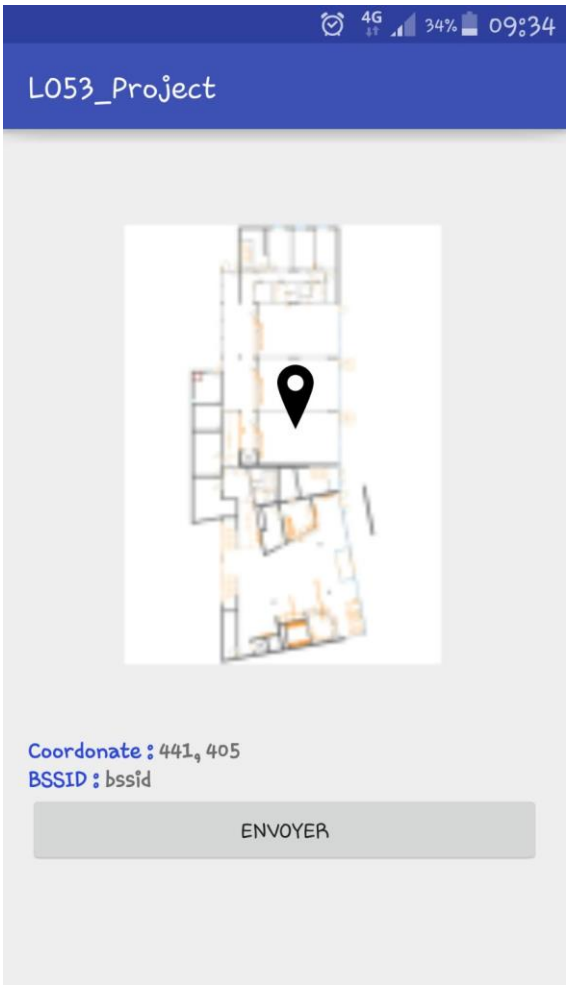
When you launch the application, you can choose to put the settings, to locate or to calibrate.

In the settings activity, we have to setup the server information such as IP server, IP port and the path of the post request. We have to set these settings before trying calibration and location, in order to know which server you want to request (either CalibrationServlet or Positioning Servlet).

CalibrateActivity of the application allows the user to calibrate the access points with the Android device.

The third part of the application is the LocateActivity which displays the default map. A request is sent to our Positioning server when we enter this activity to find the position of

the user in the room. Once the request is received by the server, a response is sent back and a pin is displayed on the map according to the locating algorithm which uses the three nearest points in signal strength.



II. Server

Calibration

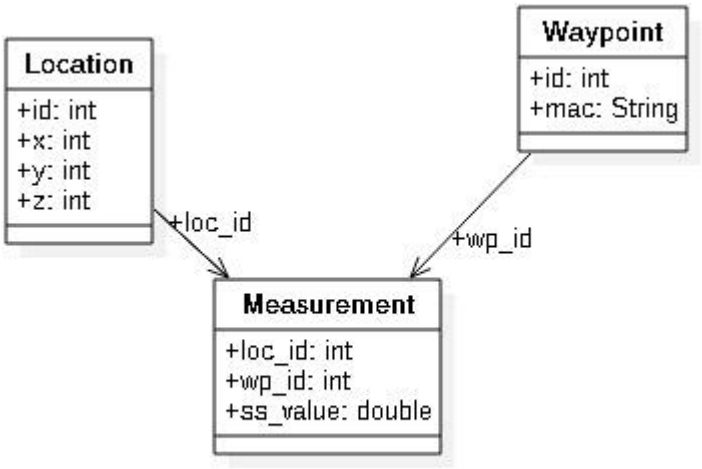
In order for the system to be accurate for the computation of the location of a mobile, it has to be calibrated first. The calibration process is going as follows.

1. The mobile sends a HTTP POST calibration request of type “calibration” to the server (via the servlet /IPSServer/Calibration/Servlet). This request also contains the coordinates (x,y,z) of the mobile device.

2. The server adds the location to the database. It also sends a broadcast request to all the waypoints.

3. The waypoints will send a (wp_mac, rssi) pair composed of the waypoint MAC address and the rssi value to the server and these data (x,y,z,wp_mac,rssi) will be added to the database.

   These calibration data in the database will be used to calculate the location of any mobile devices sending a positioning request of type “findme”.



Positioning

Now after the system has been calibrated, computing a location of a mobile device is possible. The positioning process is almost the same as the calibration request except it doesn’t add the data in the database, but instead return the location to the mobile device.

1. The mobile sends a HTTP POST positioning request of type “findme” to the server (via the servlet /IPSServer/Calibration/Servlet).

2. It relays a broadcast request to all the waypoints.
3. The waypoints will send a (wp_mac, rssi) pair composed of the waypoint MAC address and the rssi value to the server and location will be computed and sent back to the mobile under the form (x,y,z) which are the coordinates.

To compute the location, we used the tri-lateration which is a method that can find a position *P* knowing three other positions and their distance to the position *P*.

The tri-lateration method is as follows.

1. The server finds the three nearest locations stored in the database (if there are more than three waypoints) from the current mobile device according to the RSSI values. The distance between each calibrated measurements and the current measurements are calculated as described in figure. Then, the three nearest locations are the three ones associated with the smallest distance values.

$$d = \sqrt{\sum_{i=1}^{N} (ss_{ji} - ss_i')}$$

$$L = \left\{ Pj \Big/ \min_{1 \le j \le M} \left(\sqrt{\sum_{i=1}^{N} (ss_{ji} - ss_i')} \right) \right\}$$

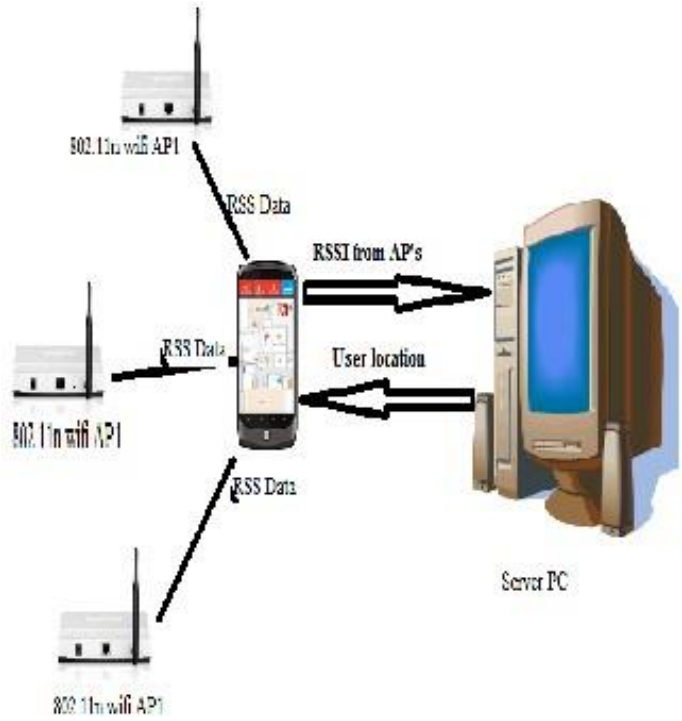
2. The server uses tri-lateration to find the approximate position of the current device. The final value *p* is the computed coordinates (x,y,z) of the current mobile device.

$$w_n = \frac{1}{d(SS_{pn}, SS) + \varepsilon} \text{ for } n = 1 \text{ to } 3.$$

$$p = \frac{w_1 \cdot p_1 + w_2 \cdot p_2 + w_3 \cdot p_3}{w_1 + w_2 + w_3}$$

III. Waypoints

In order to achieve the goal of our project, we need some waypoints which the role will be to play the intermediary between the map server and the devices that need to be geolocated. For this project, we have chosen to use need 3 waypoints.



1. Spécifications

The waypoints are of type TP-LINK N600 and they all run with the OpenWRT Operating System.

2. How they act

Globally, the waypoints have two main functions. The first function is to capture the RSSI (Received Signal Strength Indication) samples of the Wi-Fi devices that are nearby based on its own Wi-Fi interface, and the second function is to get Http requests from the map server and to answer to the requests accordingly to the need of the map server.

The map server needs the average signal strength emitted by a specific Wi-Fi device, and the waypoints calculate the average value using the captured RSSI samples gotten from the specific device.

In order to do all these operations, a program need to be written and specifically cross-compiled for the OpenWRT system that reside on the waypoints.

3. The implementation

The program for the OpenWRT have been coded in C language and some very useful libraries have been included to the program and added to the makefile in order to compile the program accordingly to our needs.

But from these libraries, two of them are the most important; those which cover the two main functions of the system:

- **Microhttpd**: it allows us to design a very small Http server in our program which will receive the map server requests.

- **Libpcap**: it allows us to write a specific code based on radiotap and its headers to catch the Wi-Fi frames or packets and to make some necessary extractions on them to get information.

  It was expected to analyze the packets by using the headers of the radiotap but due to some problems we encountered when using it we finally decided to extract the desired information with an alternative solution based on **Tcpdump**.

  Tcpdump is a common packet analyzer which allows us to display TCP/IP and other packets being transmitted or received over a network; it is also based on the **libpcap** library to capture packets.

We have divided the program in three main tasks (threads); one task that will run in background the tcpdump command on the system, another task to manage the capture of the RSSI samples from the gotten packets, and the last task to clear the outdated samples and devices based on some fixed deadline values.

The microhttpserver have been initialized in the main function and developed in a header file.

- **The capture process:**
  As soon as the program starts, the three threads are executed. The tcpdump command is first executed and the program starts extracting the information (source mac addresses, signal strength) from the tcpdump output and stores them in linked-lists. Each new mac address is seen as a new device, and the captured RSSI samples (signal strength) are matched accordingly to the device concerned.

- **The communication process:**
  The waypoint can be contacted by the mapserver in two circumstances either during the calibration process or during the positioning process.
  In both circumstances, the waypoint act the same way:

  - It first receives an HTTP request from the map server including the mac address of the device whose rssi samples are needed.

  - Then it searches the concerned device, calculates the average signal strength based on the known RSSI samples.

  - And finally it responds to the map server by providing him the information requested (the waypoint mac address, the device mac address, the average RSSI value, and the number of RSSI samples).

D. Results

In this project we had to implement a system that includes many aspects (the mobile device, the map server, and the waypoints). As result, we can tell that the project has been implemented to 90%.

We managed to make all parts of our indoor positioning system communicate with each other (Android application, Server and Waypoints).

The mobile application has been implemented and its connection to the map server has been done without any problem using POST requests.

On the waypoints side we encountered at the beginning some problems when we were trying to extract the information from the radiotap headers; it took us a non-negligible time and we then decided to use an alternative solution (the tcpdump utility) which finally helped us to get the RSSI samples from different mac addresses without any major problem.

Also, the map server actually reacts as expected; it communicates well with the mobile device during the calibration process and it also receives the RSSI samples from the waypoints.

After receiving them, depending on the request (calibration or positioning), it behaves accordingly by adding data into the database or by returning the location computed.

Below, you can see an example of a result sent from the access points. It is a string in JSON format that is returned. So it is easy to retrieve the data using a JSON parser library.

{"infosignal": [{"mobile_mac":"5c:2e:59:f5:52:2a","ap_mac":"e8:94:f6:c4:0e:f1","number_of_samples":"8","avg_rssi_value":"-35.250000"}]}

```
    Sample 4: -56.000000    deadline : 5
    Sample 5: -60.000000    deadline : 5
Device 23: 28:cf:e9:19:d6:a5
    Sample 1: -65.000000    deadline : 5
Device 24: 8c:a9:82:b9:80:f0
    Sample 1: -88.000000    deadline : 5
Device 25: 28:b2:bd:cf:0c:33
    Sample 1: -82.000000    deadline : 5
Device 26: e4:ce:8f:2d:81:da
    Sample 1: -84.000000    deadline : 5
    Sample 2: -82.000000    deadline : 5
    Sample 3: -82.000000    deadline : 5
Device 27: 7c:5c:f8:41:83:3b
    Sample 1: -87.000000    deadline : 5
Device 28: e4:f8:9c:5c:29:66
    Sample 1: -49.000000    deadline : 5
Device 29: 68:5d:43:6a:df:c4
    Sample 1: -85.000000    deadline : 5
Device 30: dc:85:de:79:6b:6b
    Sample 1: -86.000000    deadline : 5
    Sample 2: -86.000000    deadline : 5
    Sample 3: -85.000000    deadline : 5
Device 31: b4:8b:19:0e:7f:61
    Sample 1: -40.000000    deadline : 5
    Sample 2: -40.000000    deadline : 5
Device 32: 00:1b:90:0a:ae:17
    Sample 1: -85.000000    deadline : 5
Device 33: e4:58:b8:2e:89:61
    Sample 1: -52.000000    deadline : 5
    Sample 2: -49.000000    deadline : 5
    Sample 3: -48.000000    deadline : 5
    Sample 4: -50.000000    deadline : 5
Device 34: e8:94:f6:bb:bd:9e
    Sample 1: -40.000000    deadline : 5
```

In the figure above, these are the RSSI values captured by one of the access point of our system along with the corresponding MAC address.

E. Perspectives

The project has been implemented as it was described in the statement. But it can still be optimized in several ways:

- The map on the Android application is not precise. Indeed, we cannot perform a zoom on it yet. Therefore, we could implement a zoom functionality for the map in the mobile application.
- We also haven't adjusted the coordinates to the map perfectly so you could see that some of the coordinates are invalid (out of the map).
- This application has been developed for testing purposes. So at the moment, the user has to put some settings like the servlet address for POST request. Nevertheless, these settings should be setup automatically when sending a request to the corresponding servlet.
- Sometimes, the RSSI values are nil even if the mobile device is in the range of the waypoints. This could be corrected by doing more requests to the waypoints until it receives a satisfying RSSI value.
- Implementing a real-time map on the mobile application that will be automatically updated based on the user's movement would be a great improvement as well.
- Finally, we could replace the waypoints with more powerful ones which could have a more efficient wireless card to cover a larger area.

Conclusion

The implementation of an indoor positioning system based on access points is not very famous and it needs a considerable range of knowledge both in programming and network domains.

This project was a great opportunity for us to put in practice the concepts we acquired in the LO53 course. We discovered that the implementation of such a system pass by some steps and each step must be carefully processed in order to have a stable system for an efficient geolocation.

Sources

1. System and method for determining location of a wi-fi device with the assistance of fixed receivers (Javier Cardona, Frédéric Lassabe, Alejandro Herrera)
2. http://yuba.stanford.edu/~casado/pcap/section1.html
3. http://openmaniak.com/fr/tcpdump.php
4. https://www.gnu.org/software/libmicrohttpd/
5. http://www.journaldev.com/7148/java-httpurlconnection-example-to-send-http-getpost-requests
6. http://theopentutorials.com/tutorials/android/http/android-how-to-send-http-get-request-to-servlet-using-apache-http-client/