Wi-Fi positioning System

Frédéric Lassabe
Université de Technologie de Belfort-Montbéliard

2012 Spring

Contents

1 Requirements

1.1 Virtual Machine

For this project, you work on a Debian GNU/Linux virtual machine provided in the folder `/vm` of the lab room computers. To import the VM, you have to follow the following steps:

- in the "File" > "Preferences" menu, go under the general settings section and set the default VM folder to `/vm/<you login>`,
- in "File" menu, use the "Import virtual Appliance" and choose file `/vm/lo53.ova` as source,
- You can now start the Debian VM and log in with user "lo53" and password "lo53". User "lo53" is in the sudoers so that you may use "sudo" command.

WARNING! Do not forget to save your data at the end of each lab session. Since the virtual machines might be erased by maintenance procedures, you could otherwise lose all work previously done.

1.2 Skills

This project considers that you have:

- good knowledge and practice of C programming language,
- good knowledge of Unix-based operating systems,
- good knowledge of Java programming language,
- a good understanding of HTTP protocol and IEEE802.11 standard.

2 Introduction

During lab work, you will have to build a full Wi-Fi based positioning system. This system is based on 3 parts:

- Wi-Fi access points
- Android terminals
- Positioning server

It is restricted to a LAN usage to be able to map MAC addresses on IP addresses without relying on the phone to provide its own MAC address.

You should divide the work the following way:

- first two weeks: AP developpment;
- two following weeks: server developpment;
- last two weeks: mobile developpment.

2.1 System definitions

The **user** is the person carrying a mobile terminal. A **mobile terminal**, a.k.a. **mobile device** is a handheld computing device, e.g. an Android smartphone. The **positioning server** is a computer providing algorithms to locate a mobile terminal as well as interfaces to interact with mobile terminals and access points. **Access points** are the wireless access points which provide Wi-Fi network access to mobile terminals as well as RSSI measuring capabilities.

System	Abbr.	Software
Mobile terminal/device	MD	Android package
Positioning server	PS	Tomcat Application server PostgreSQL database
Access Points	AP	WRT54GL with OpenWRT firmware

Table 1: Definitions of terms and abbreviations.

2.2 System overview and operation

There are two phases to consider in this system. First one is the offline phase, during which a RSSI map is built. Second one is the online phase, during which mobile devices are located.

2.3 Offline phase

The offline phase requires a mobile terminal with a map of the building, AP deployed inside the building and the positioning server. The procedure to create a point of the RSSI map is the following:

1. The user selects his location on the map displayed on the mobile device screen. The point coordinates are calculated based on the map scale (previously defined by the user). Then, the user pushes the "Measure" button. When the button is pushed, a burst of packets is broadcast during 500 milliseconds.

2. Each AP in range measures and stores RSSI values.

3. Immediatly after the packets burst, the mobile device sends a message to the server. Message is "MEASURE;X;Y;mid" where the X is the point x-coordinate, Y is the y-coordinate and *mid* is the id of the map displayed on the mobile device.

4. when receiving the message, the server looks for the mobile device MAC address in its ARP table and sends a message to all the AP: "GET;X;Y;mid;ZZ:ZZ:ZZ:ZZ:ZZ:ZZ" where the ZZ are the hexadecimal representation of the mobile device MAC address.

5. The AP look in their RSSI list. If records are present for the required MAC address, they send "RSSI;X;Y;mid;ZZ:ZZ:ZZ:ZZ:ZZ:ZZ;AP:AP:AP:AP:AP:AP;val" where:

   - X, Y, *mid* are previously defined,
   - *AP:AP:AP:AP:AP:AP* is the hexadecimal representation of the AP MAC address,
   - *val* is the average RSSI value based on RSSI samples on the last second. Be careful when averaging RSSI values since dBm are not linear, you have to convert values in mW, then average the mW values and finally convert it back to dBm.

If the MAC address is not in the list, then *val* equals -95.

6. Upon reception of the AP responses, the server stores the values in the RSSI map.

The RSSI map, as its name says, maps locations on RSSI values. A location is defined by a map identifier, and a 2D-coordinate. Each location is associated to one RSSI value for each AP in the system. Whenever an AP is out of range of a given location, its RSSI value is considered as equal to -95 dBm.

2.4 Online phase

During this phase, the steps are the following:

1. The mobile device sends a burst of packets immediatly followed by a message LOCATE to the server.
2. The AP measure the RSSI values of the packets.
3. Upon reception of the LOCATE message, the server requests the RSSI samples from the AP with a message "GET;XX:XX:XX:XX:XX:XX" where the XX are the MAC address bytes in hex. representation.
4. the AP send back a response "RSSI;XX:XX:XX:XX:XX:XX;AP:AP:AP:AP:AP:AP;val" (see offline phase for definitions of the fields).
5. By comparing the values returned, the server computes the location closest of the AP measurements.
6. The server returns the coordinates and map identifier to the mobile device.
7. Mobile device displays its location on the map.

You have to take into account the lack of global clock to manage the RSSI responses from APs: when you send the message, you have to wait a certain amount of time – while getting the responses – before answering the mobile device.

In the following parts, requirements and guidelines are provided for developping each component of the system.

3 Access Points

The access points require to develop a RSSI measurements software. You have been provided a virtual machine image **lo53.ova** with a cross compilation toolchain for OpenWRT operating system on MIPS architectures.

The access point shall keep a list of mobile devices identified by their MAC addresses. The list is maintained with a sliding window of 1 second. When a frame has been received more than one second ago, it is erased from the list. In parallel from keeping this list, the AP RSSI measurements program must listen to messages from the positioning server.

The positioning server has one message, named GET. It is sent through a TCP socket and is coded with the following text string: "GET;XX:XX:XX:XX:XX:XX" where the 6 two-characters parts are the hexadecimal values of a MD MAC address to find in the list. The AP answers by sending back the average RSSI value for the MD required. If there is no record available, its reply shall inform the server that there is no value.

The AP scans and measures RSSI values with libpcap library and the *prism* header. Its replies to the server are based on UDP messages.

What you have to develop for the AP:

- a linked list containing the RSSI samples and their expiration date. The expiration date determines when to drop a sample from the list,
- a monitoring pcap handler to get RSSI samples,
- a listening UDP socket to get the server requests for RSSI information,
- use the same UDP socket to send responses to the server,
- a timer to delete outdated RSSI samples.

You might use POSIX threads and semaphores to get all the components to work together. Steps to get the system working:

- Check the compilation with a hello world program compiled with the cross-compilation toolchain and copied (use scp) to the WRT54GL,
- Get used to libpcap. For instance, you write a draft program which opens the prism0 interface with pcap, then loops on the pcap device listening and displays all frames MAC addresses and RSSI. At this step, do not store anything nor try to communicate with the server.
- Modify the previous program to store the values and display the liste each second. You also remove all values older than one second.
- Finally, add a thread to manage UDP communications with the server. Don't forget to use a semaphore when accessing to the device RSSI records list.

The remaining components (server and mobile device) shall be available soon.

4 Positioning Server

The positioning server embeds 3 servlets:

- a servlet to manage AP RSSI responses,
- a servlet to manage mobile devices positioning requests,
- a MBean or a ServletContextListener (or any other viable solution) to listen APs responses on UDP port.

4.1 RSSI database

We keep a simple database with one table for locations, one table for access points, one table for RSSI measurements (floating point values with average and standard deviation values) which links to locations and access points, a temporary table for RSSI readings in the APs responses, and a table to store pictures of the maps.

Table	Field	Type
Location	id	Integer (note: auto increment, pkey)
	x	Double
	y	Double
	map_id	Integer
AccessPoint	id	Integer (note: auto increment, pkey)
	mac_addr	Varchar(18)
Rssi	id_loc	Integer (ref. id in Location)
	id_ap	Integer (ref. id in AccessPoint)
	avg_val	Double
	std_dev	Double
TempRssi	ap_id	Integer (ref. id in AccessPoints)
	client_mac	Varchar(18)
	avg_val	Double
Maps	id	Integer (note: auto increment, pkey)
	description	Varchar(100)
	px_width	Integer
	px_height	Integer
	meters_width	Double
	meters_height	Double
	content	binary (map picture)

4.2 Managing mobile device positioning requests

This servlet gets the request, searches the mobile device MAC address in the server's ARP table (read in /proc and filter by IP address). Based on the MAC address, the server sends a request for RSSI samples to the APs. After sending the request, it sleeps for 500 milliseconds. Then, it gets the RSSI data and determines the location based on a simple *nearest point in signal strength*.

The process listening to the APs responses is in charge of storing temporary data into the database where other servlets can read it. Other servlets are in charge of cleaning records when they have been used.

4.3 Managing mobile device RSSI map calibration

This servlet is similar to the previous one except that:

- it receives the coordinates of the location (sent by the UE)
- it stores data from the APs in the RSSI map.

4.4 Managing AP RSSI responses

This component of the server is a background process listening for access points RSSI responses on a UDP socket and storing the values in the TempRssi table of the database.

A libpcap C language sample code

```c
#include <pcap.h>

/* Content omitted */

struct ieee80211_header
{
  u_short frame_control;
  u_short frame_duration;
  u_char recipient[6];
  u_char source_addr[6];
  u_char address3[6];
  u_short sequence_control;
  u_char address4[6];
};

/* Content omitted */

pcap_t * handle = NULL;
struct prism_header *ph;
struct ieee80211_header * eh;
struct pcap_pkthdr header;
const u_char * packet;

/* Content omitted */

handle = pcap_open_live("prism0", BUFSIZ, 1, 1000, errbuf);
if ( handle == NULL ) {
  printf("Could not open pcap on interface\n");
  return -1;
}

/* Content omitted */

while ( 1 ) {
  packet = pcap_next(handle, &header);
  if ( ((unsigned int *) packet)[0] == 0x41 ) {
    ph = (struct prism_header *) packet;
    eh = (struct ieee80211_header *) (packet + ph->msglen);
    // Check if FromDS flag equals 0
    if ( (eh->frame_control & 0xc0) == 0x80 ) {
        /* Do something with (ph->rssi).data */
    }
  }
}

/* Content omitted */
pcap_close ( handle );

/* Content omitted */
```

B pthread C language sample code

```c
#include <pthread.h>

/* Content omitted */

static void * my_func ( void * args )
{
  int max = (int) *args;
  int i;
  void * ret = malloc ( sizeof ( int ) );

  /* If sharing memory with main process,
     use inter-process synchronization */
  for ( i = 0 ; i < max ; ++i ) {
    (*ret) += i;
  }
  pthread_exit ( ret );
}

/* Content omitted */

pthread_t comm_thread;
struct thread_params tp;
int * my_ret;

/* Content omitted */

pthread_create ( &my_thread, NULL, my_func, (void *) &tp );

/* Content omitted */

pthread_join ( my_thread, void (**) &my_ret );

/* Content omitted */
```

C Listening UDP socket C language sample code

```c
#include <string.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <unistd.h>

#define PORT 12345
#define MY_IP "192.168.1.11"
#define BUFFER_SIZE 5000

/* Content omitted */

int skfd;
struct sockaddr_in addr, remote_addr;
char buffer[BUFFER_SIZE];
size_t addr_size;

/* Content omitted */

addr.sin_family = AF_INET;
addr.sin_port = htons(PORT);
inet_pton(AF_INET, MY_IP, &addr.sin_addr);
memset(&addr.sin_zero, '\0', 8);
skfd = socket(PF_INET, SOCK_DGRAM, 0);
if ( skfd == -1 ) {
  printf ( "Could not open skfd\n" );
}
if ( bind ( skfd, (struct sockaddr *) &addr,
            sizeof ( struct sockaddr ) ) == -1 ) {
  printf ( "Error when binding socket\n" );
}

/* Content omitted */

if ( recvfrom ( skfd, buffer, BUFFER_SIZE - 1, 0,
                (struct sockaddr *) &remote_addr,
                &addr_size ) > 0 ) {
  /* Use data from buffer
     (now remote_addr contains addr info on remote device) */
}

/* Content omitted */

close ( skfd );

/* Content omitted */
```

D RSSI list

You have to use a double linked list with the following prototypes:

```
typedef struct _RssiList {
  struct timeval rl_date; // Expiration date
  int rl_rssi_value;
  struct _RssiList * rl_next;
} RssiList;

typedef struct _DeviceList {
  char dl_mac_address[6];
  RssiList * dl_rssi_list;
  struct _DeviceList * dl_next;
} DeviceList;
```

You have to define at least the following functions:

```
void add_device ( DeviceList ** l, char mac_addr[6] );
void clear_device_list ( DeviceList ** l );

void add_rssi_sample ( DeviceList * l, int rssi_value );
void clear_rssi_list ( DeviceList * l );
void delete_outdated ( DeviceList * l, struct timeval current_time );
```

E WRT54GL Broadcom Wi-Fi monitoring mode

To get RSSI readings for all frames received by the AP, you need to set its Wi-Fi interface to monitoring mode. Setting the monitoring mode requires one simple command:

```
root@openwrt$ wlc monitor 1
```

After issueing this command, a new interface appears when listing interfaces. This new interface is labelled "prism0". It mirrors all frames received. Each frame is completed by a prism header. This header makes frames of the prism interface useless for applications but extremely useful for developpers and administrators who want to monitor surrounding Wi-Fi signals.

The "prism0" interface has to be opened with libpcap to get all the information required from the frames.