Wilocator (IPS)

Senior Design Project

Group Members

Rodney Dulin

Davion Teh

Kelvin Yap

Advisor: Professor John Kapenga

Department of Computer Science

December 20th, 2014

Abstract

We have created an IPS, Indoor Positioning System, which we call WiLocater. The project started as an upgrade to an open source IPS that our group found online. After a lot of research, many problems, and a lot of debugging we decided to scrap the upgrade process and develop our own system. The system is developed with the user in mind. A user with little to no background of technology can use it, is easy to set up, and is open source. There are two pieces to our system. The first is the server, programmed in C sharp and the second is the client, programmed in java and runs on an Android device.

We have created the IPS to get around inside a building. Someone with a mobile device and WiLocator can download a floor plan of the building and the app will give the user their location. No need to look around the building for a map to find where they need to go. Also it can be used by multiple users and it will work as fast as the computer and wifi connection will allow. There can be many people connected within a big building and as long as they have a connection the system will work.

Wilocator also has an emergency feature. This feature could be of use in many public places like schools, malls, or any entertainment venue. If something happens within the building and the rest of the building should be notified. An administrator can notify them with a tap of a button and an alert will be sent out to every device connected to the server.

The server is self maintaining. There is no need for someone to monitor it. The only time someone will actually interact with it will be set up. Admin can backup and load the server when needed.

Acknowledgements

This project was designed and developed for Dr. John Kapenga, Computer Science Professor at Western Michigan University. We were provided with access point units and android tablets while developing this project. Dr. John Kapenga also provided his advice and feedback from the beginning until the end of the project development.

Table of Contents

Introduction

GPS, Global Positioning Systems, are not an accurate form of indoor position tracking because of obstructions to the signal.A solution is needed to track indoor positions. In brief context of how Indoor Positioning generally works (since there is more than one approach), imagine that there are two people, A1 and A2, standing at the end of a hallway which acts as an access point. Whenever someone enters or exits the hallway, they will have to pass by A1 or A2 that are standing next to the entrance/exit of the hallway. A1 or A2 will then report the details of the person that had just passed by to the server, and the server will then know the location of the person because he has just recently in proximity of A1 or A2. This is one of the approaches to create an IPS, which is called “Choke Point Concepts”.

We have created a program which we call WiLocator. It was developed with the goal of providing the users position with at least room level accuracy. While not needing to do a training and a setup phase, which is very time consuming and costly. WiLocator uses a fingerprint system, meaning it does not provide geographical coordinates and instead it provides “symbolic identifiers”.

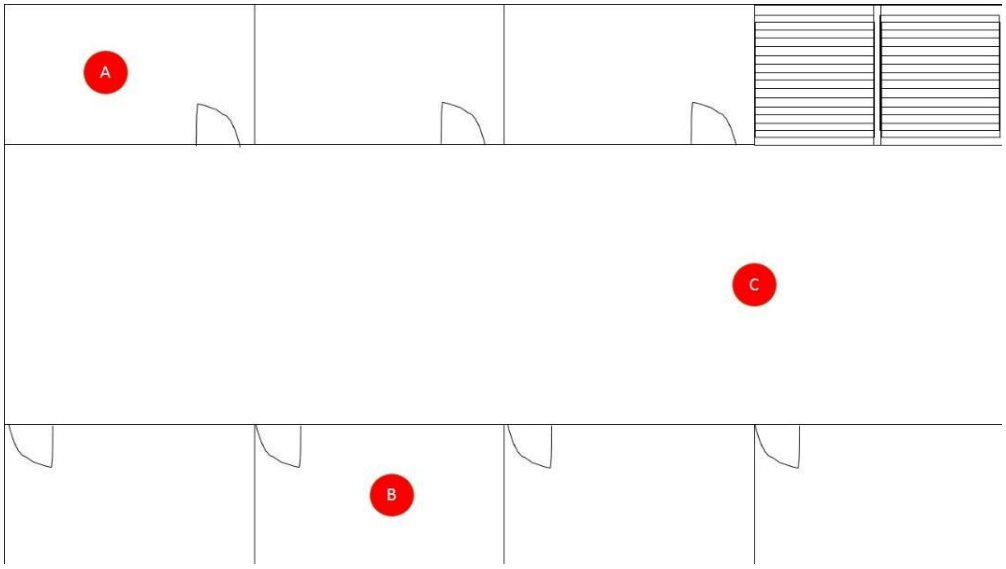


Figure 1- Location are determined by Symbolic reference instead of Geographical coordinate

Further details such as IPS concept, WiLocator's approach, and what we are trying to do will be further discussed in the main part of this report. The system is based on 3 different parts:

- Wi-Fi Access Points [AP]
- A position computing server
- An Android device

There are two phases to consider in this system. The first phase is called offline (calibration) supposed to build a RSSI, Received signal strength indication, map. The second phase is called online (positioning). This phase will locate the device in an indoor environment (room). The whole project was built on a Windows Machine to allow us to develop with all the environments and the IDE, interactive development environment, needed.

We used the Android software development kit (SDK) with the Android developer tools (ADT) plugin for Eclipse to create our different android application. The server is programmed using C# with Microsoft Visual Studio as an IDE.

Background

We are living in the age where people heavily rely on navigation technology for travelling purposes, be it long or short distance. The rise of GPS technology, its high precision and accuracy to be able to guide travelers correctly to their destination has sparked interest and demand for more of this technology. However, the downside of GPS is that it is rendered useless when the device is under some sort of cover like buildings or tunnels. After many attempts to solve this issue, a new system has been discovered to cover what GPS could not do. The new tracking system is called Indoor Positioning System.

The reason why GPS does not work indoors is that solid matter blocks the signals it tries to send to the satellite. IPS on the other hand does not rely on satellites; instead it relies on sending signals to multiple WIFI routers or bluetooth devices then receives its current location from the server after some geological math computation. This new discovery may sound easy, but in practice it is currently very difficult to efficiently and accurately track the position of a device with the current system design.

This system requires heavy infrastructure deployment of access points or bluetooth devices to even "try" to achieve room level accuracy.

This report will explain our project in detail including how we designed the system, what we used to design, what research we did, and how we tested the system.

Spikes

Spikes are tests and for the start of this project, the spikes will be just that. The first thing we had to do was some research. In order to do this project we had to have a little background in android and indoor positioning systems.

- determine if there were any other products or programs that would work like Redpin and if they would be a better fit for our project.
- look at what version of android Redpin was currently using and what version of android was the newest and how difficult it would be to update
- we had to determine how indoor positioning systems worked, and particularly how Redpin worked

We also had to test in order to see if Redpin worked in its current state, since the project must work to be upgraded. Test if the server communicates with the program on an Android device and determine which versions of android will and will not work. Lastly which PC operating systems will work.

If the original Redpin worked in its current state, the project could then proceed, otherwise we would have to continue to look at and debug code. We tested the system and could not get it to work correctly, so we had to research further:

Phase 1

- What version of android can Redpin run on?
    - android 2.2
- What language is used to program the interface?
    - XML
- Is the interface supported on all devices that android will run on?
    - smartphones and tablets

Phase 2

- What OS could the server be run on?
- Can the device communicate with the server?
- What information are being sent/received?
- What information should be stored on the server?
- What information should be stored in the device?
- Is it possible to upload an image to the server from the device?
- Can the server convert uploaded image to usable map?
- What happens if a map already exist?
- How to validate the maps?
- Can the device compute its location?
- How precise are the computations?

We realized that the way Redpin calculated the users position would not work and decided to create our own IPS. This decision included more research, in which we had to do:

- How to create the GUI?
    - Target: Java Canvas
- What will the GUI look like?
- Is it possible to convert an image to a usable map?
- What type of image file formats will be supported?
    - Ex: png, jpeg, bitmap, gif
- What are the minimum/maximum image sizes will the program support?
- How to validate the image file?
- How to obtain data from wifi access points?
- Is it possible to uniquely identify access points with their MAC address?

After we have done the research we can get different pieces of the system programmed and testing can begin.

Design Decisions

We started our project knowing we wanted to develop an open source IPS. We did some searching and found Redpin, an open source IPS that ran on Android and IOS. It needed to be upgraded and we wanted to add an emergency feature, which would let any user click an emergency button and it would alert other users of an emergency in the building.

After some testing and research we decided that Redpin had a bug that rendered the program useless for our needs. Redpin calculates position by dividing the users distance from all APs by the number of APs, which in our case does not work, because we want our system to be as user-friendly as possible and with this method removal of an AP will force the user to recalibrate all fingerprints.

We have done a lot of research and have the knowledge to develop our own system. Our next decision was to use a system of calculating position with trilateration. With Trilateration, which is similar to triangulation, we would need coordinates of all APs. We don't always know where these are in a building and we didn't want to rely on coordinates. We realized quickly that this was not an option for our system, because we would like the system to be used anywhere and there places where we might not know where the APs are.

The method we used to calculate a device's position is called data comparison. It is a lot easier to calculate the user's position with this method as well.

We choose to write the server in C-sharp as a server over Ubuntu server, because we are more familiar with it and our client allow us to do so. Next as for the client, it is written in java so that the user can run the program on all different android platform.

Stories

IPS Alternatives

There are many solutions to indoor positioning. Many of them are not open source and can be very expensive. We live in a world where technology is available and changing daily. We did research in this subject and decided we needed something open source, used on an open source platform preferably Android, and something user friendly. At this moment in time, the indoor localization system is one that is currently in demand by most businesses and consumers. In depth researches show that there are several existing solutions out there in the market. One of them would be Navizon Indoors to Track (l2T). It enables tracking of the user with accuracy of about 2-5 meters. Navizon provides a SDK for developers to customize the client side apps for both android and iOS. The training of the map is done by moving across the site, recording signal at specific locations displayed by the Navizon application. The users will be able to use their web service to query Navizon Indoors' server to obtain its position.

Another example would be Estimote beacons. Estimote used the iBeacon that is developed by Apple, built into their operating systems and hardware. Users would need to have a bluetooth connection to these beacons. The iBeacon devices are deployed around a room to map a space using a quick orientation process using the app, with directions provided by the software itself. The solutions by Estimote are expensive because it requires the purchase of the beacons hardware.

Q-Track however does not require wifi and instead it uses the distance-dependent difference in phase to detect a user's location. It does not require the scanning of wifi signal strength to calculate the distance between their transmitter and receiver. Q-Track claims that it can provide a

user's location up to submeter accuracy by detecting their transmitter tags with respect to fixed receivers. These receivers can measure electric and magnetic fields despite the structural elements of a building.

Redpin IPS

Having Indoor positioning system as our developing experiment target, our group has been searching for information regarding IPS. We found an open source project called "Redpin", which the goal of its development is to create a new concept of an IPS system. After checking through its documentation, it is a form of IPS and since it is open source, we decided to use this as our main experimental development.

At first we wanted to try it out on our mobile platforms to test if it worked. However, we were unable to get it running for quite sometime (a week plus) for unknown reasons. After some further research for possible solutions, we have found that Redpin is developed to work only for Android 2.2 (not earlier versions, nor later versions). This makes us set our first objective to first update it to a version of android that is usable now, which is Android version 4.3.

When we tried to build the code in our IDE (Eclipse with Android Development Platform), the program itself could not be built even though we have not touched anything, it seems to contain errors that we could not comprehend. So it was stuck for about 2 weeks while we figured out what these errors might be. We were almost going to drop this project, because if we cannot even get the code to build it might be the case where the code does not even work in the first place.

After a few weeks of trying, we managed to find the reason behind it. We noticed that there was missing source code in order to build this Redpin project. We found that besides the Redpin

Android code files, it requires also the Redpin Core files and the Redpin Server to be on the same build path.

At the same time, we were handed a few android tablets of version 2.2 and we tested the Redpin project on them, the result was that it managed to run, but is not working, because:

1. It could not connect to the provided redpin server
2. there was no GUI to mention if it's working

We then had to first update the program to Android version 4.3, then setup a server and check to see if it could communicate with the server, then lastly setting up access points and check if it could communicate with the access points and make sure device, access point, and server are working. The positioning server where the databases are all re-created.

Code needed to be added or removed when upgrading. For example after upgrading to Android version 3.0 and above it needed an action bar added including a menu button. Tablets and some phones do not include a menu button and will need this to get to a menu of options

After modifying a few lines in the version code section, we have managed to get Redpin installed in a version 4.1 and 4.3 android device. To see if the original function still works, we tried uploading an image to the server from the device and it works fine, but another attempt to reupload the same image cause the device to crash, our hunch tells us that it might be a missing method to handle "image validation".

The other errors we found is the GUI, Graphical User Interface, were that it does not support a zooming function, doing so causes the application to crash. After finding these errors and other bugs we decided to scrap the upgrade process and create our own system.

An Android client that would communicate with a server was the ultimate goal. So we

decided to program our Android client in java using the Eclipse IDE. We also decided to use an embedded server programmed with c# as our server, because most users know Windows and possibly have another windows machine to act as the server. So our next step was to write code.

Constraining Factors on the Project

Ethics

During the project development, we used the Internet for guides on how to write code for the client and server program. We had reference and gave proper credit for code used in our software.

Manufacturability

Since this project is a proof of concept that will be delivered to our client, there is no requirement hardware that is needed to produce.

Political

Political factors do not affect the project since the project is a proof of concept for internal use only by our client. If this project was designed for public usage, political constrain should be put into account during development.

Sustainability

This project will include proper documentation for the Server and Client program. The client can modify our proof of concept project to be implemented for the next release. This documentation will include API for both server and client. Proper comments and documentation are written on different parts of the software code for better understanding. No devices are created for this project, any different type of android platform can have the APK file installed and start running. As for the server, an executable file was created to run on windows system.

Implementation

Our IPS was formed based on 4 different parts combined, the parts are **Embedded Database**, **Fingerprint**, **Fingerprint setup & calibration**, and lastly **Locating device via data comparison**.

Fingerprint

A **Fingerprint** is a **Symbolic Identifier** based on the **Received Signal Strength Indication** (**RSSI)** of a location. An RSSI is an indication of the power level being received by the antenna. In our case, the clients device will be the antenna measuring the signal strength of access point in the surrounding of the device. To setup a fingerprint, a client will need to scan it's current location such as inside a room for RSSI. After obtaining the RSSI of it's current location, the client will need to send the RSSI information and also the client's current location (current location of user is determined by where the user indicate it's location on the map) to the server to store the fingerprints details. The fingerprint on the server contains the following information:

1) Fingerprint name (string)
2) Fingerprint's coordinate (int x,int y)
3) RSSI memory management (Stack of int)
4) RSSI information of the fingerprint
    a) Associated Mac Address (hashtable of string)
    b) Signal strength indication of Mac Address (Array of short)

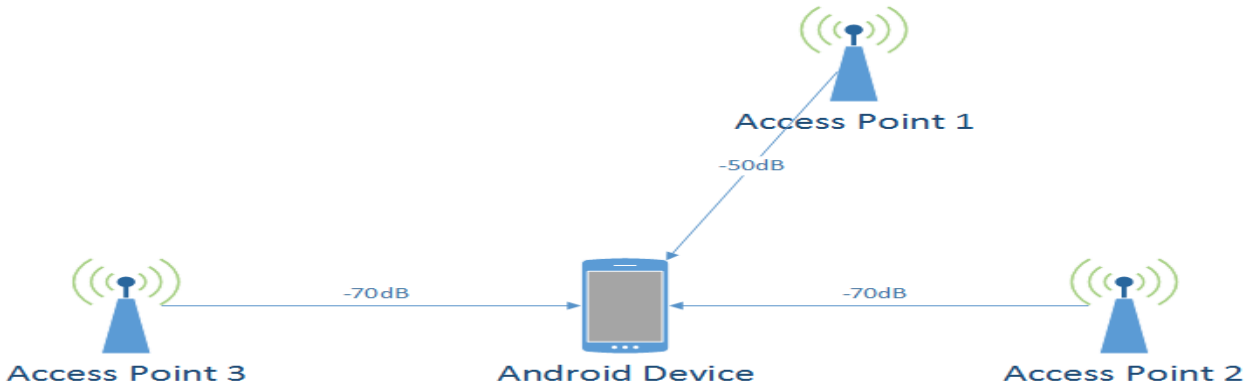


Figure 2 - Device obtaining RSSI information from surrounding's AP

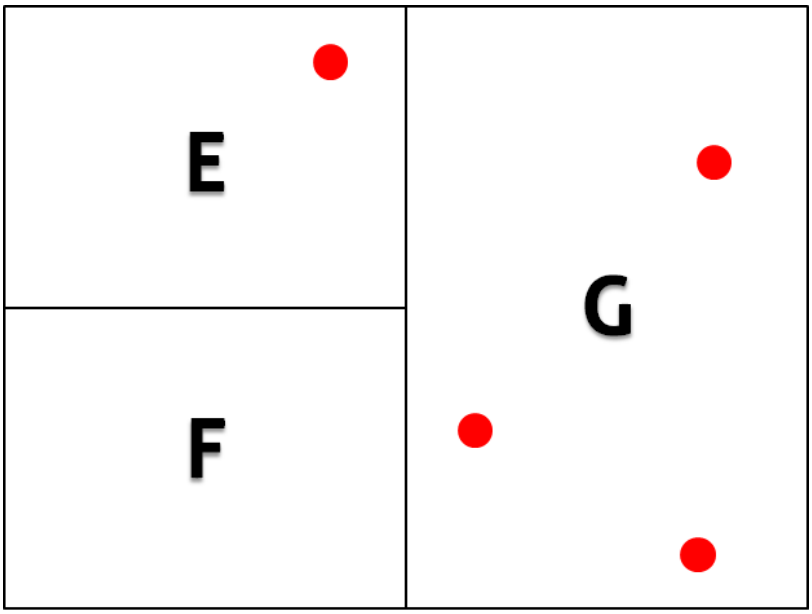


Figure 3 - Example of a building with 3 fingerprint E, F and G.

In the figure above, the red circle locations are determined by **comparing fingerprint's RSSI** with the red circle's RSSI (discussed later). This setup will allow the system to compute the device's location with at least **room-level accuracy** precision.

Embedded Database

Embedded database is a database that is included inside the program, meaning that one will only need to install the program for it to be fully functional instead of requiring to install the program and an external database. Although it may not be as efficient as a full external database, but since we're just attempting a new concept, a small embedded database is enough for our concept testing purposes.

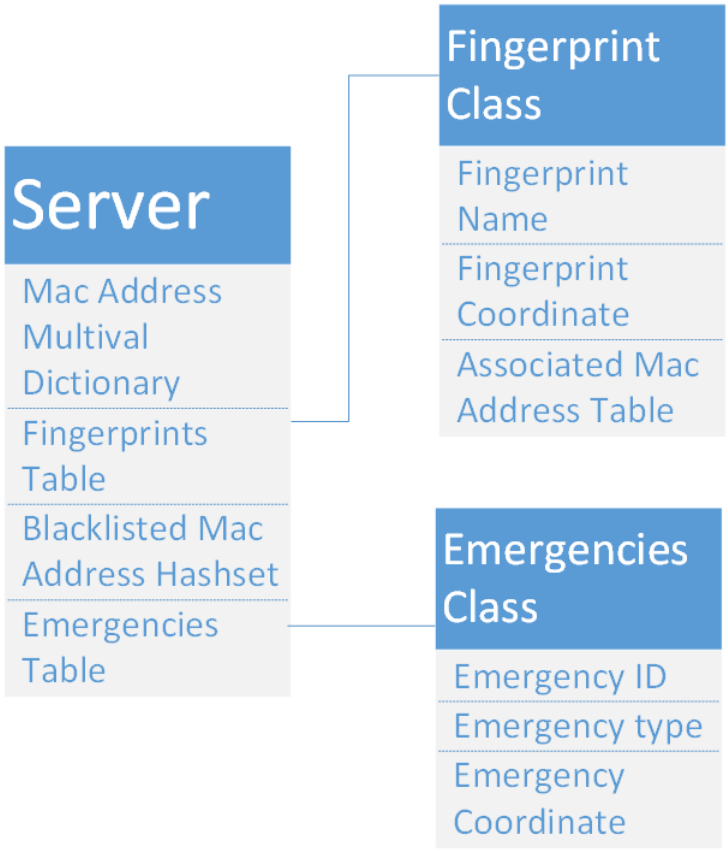


Figure 4 - brief overview of the server's embedded database

The embedded database consists of multiple types of **icollection** tied together to form a database that could retrieve fingerprint's information based on mac address given or vice versa. The icollection used are (see next page)

Multi-value Dictionary

a) Key: Mac Address (string)

b) Value: Associated Fingerprint name (string)

2) Dictionary (Fingerprint)

a) Key: Fingerprint name (Fingerprint name)

b) Value: Fingerprint object (Fingerprint class)

3) Dictionary (Emergencies)

a) Key: Emergency ID (int)

b) Value: Emergency object (Emergency class)

4) Hashset (Mac Address Blacklist)

a) key: Mac Address (string)

Fingerprint Class

1) Dictionary (Mac Address)

a) Key: Mac Address (string)

b) Value: Index of Mac address's RSSI (short)

2) List (Mac Address RSSI)

a) Min-Max RSSI for each Mac Address (short)

The server has been programmed in such a way that any creation of new fingerprints through the **parser()** method will **guarantee** the fingerprint to be added with **no duplicates**. The server is also programmed to be able to handle multiple clients attempting to create or calibrate fingerprints at the same time since the **writing process** are **thread safe**.

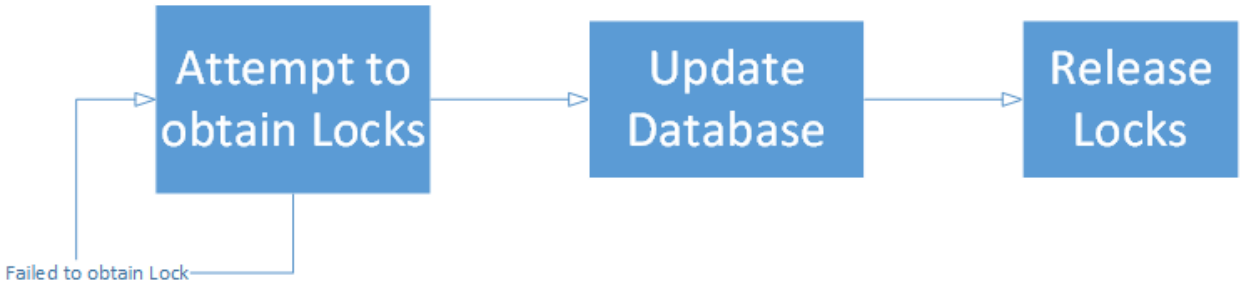
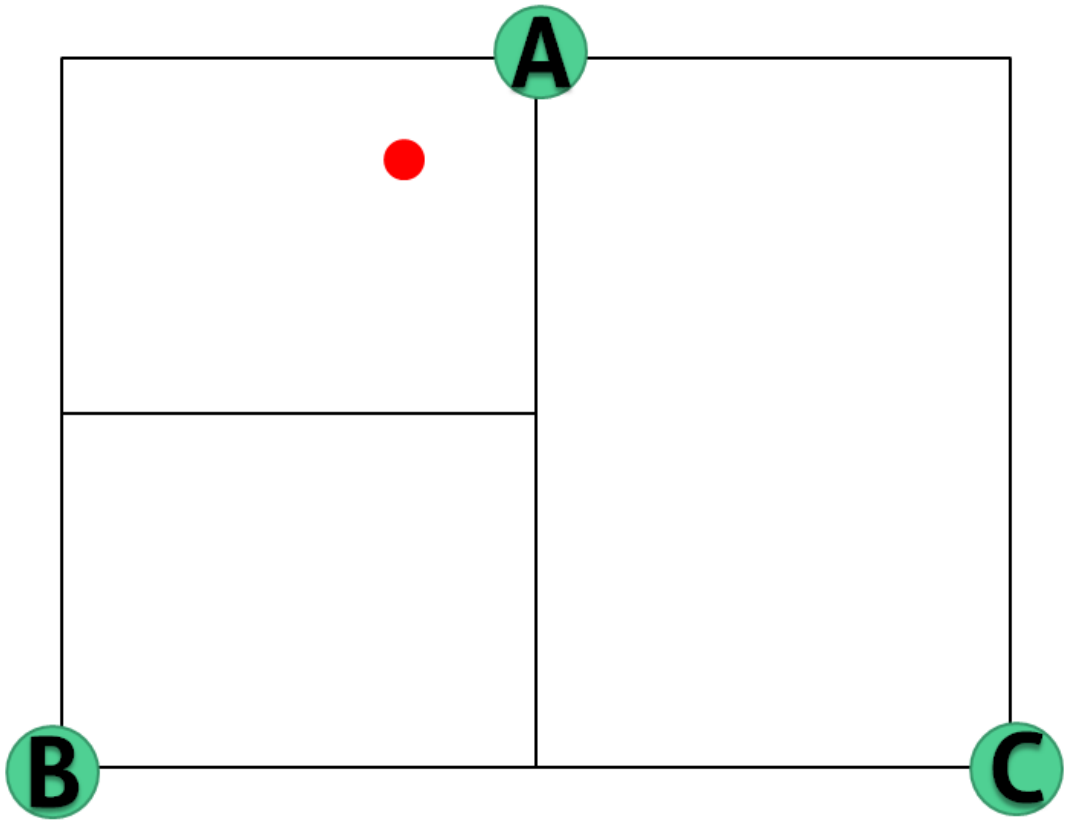


Figure 5 - Database update process

All **reading processes** are done **concurrently** with the writing processes as any failed attempts to read the required information could be repeated immediately after (**Note**: if a read is required for the update process, the update process have to obtain the locks before it could proceed to do anything else).

Fingerprint Setup

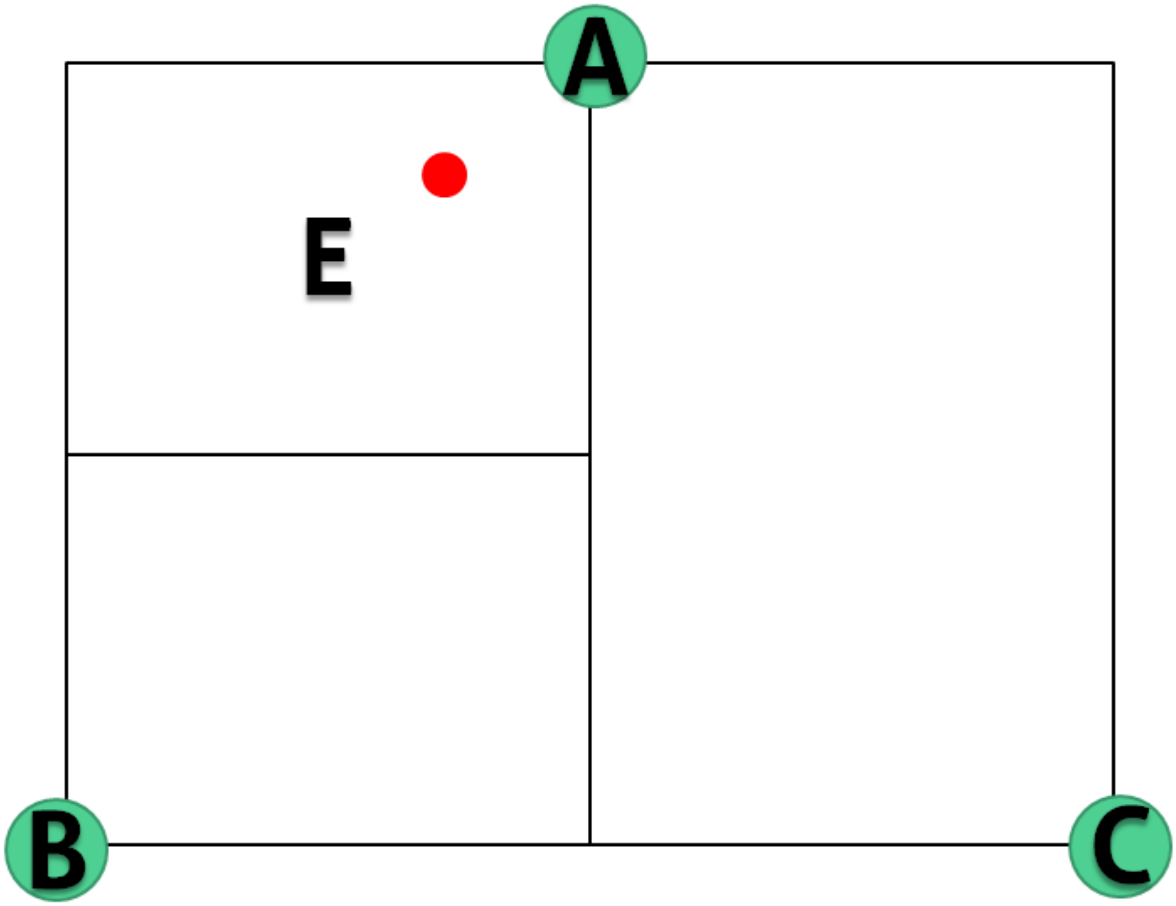
As stated in the fingerprint section above, a fingerprint has to be setup and calibrated before the system can do anything related to locating a device's location. Both the client and server have to be running and connected to each other before proceeding to the setup process. The setup process is done using the following steps below:



1) Stand inside a room (this example explains the fingerprint setup for a room, which could be applied the same to setup a fingerprint at a hallway or open space) and then scan for RSSI information for that room.

Admin	Fingerprint
Add Fingerprint	Min < AP < Max
A = -50	-50 < A < -50
B = -60	-60 < B < -60
C = -70	-70 < C < -70

2) After scanning the RSSI of that location, the information will be sent to the server and the server will store the information as a Fingerprint.



Fingerprint Calibration

After the fingerprint creation process, the user could then proceed to the fingerprint calibration process (attempt to calibrate an nonexistence fingerprint will produce an error message instead). Next, the user will need to stand in a different location of the room and then re-scan and submit the RSSI in these **different locations within the room**.

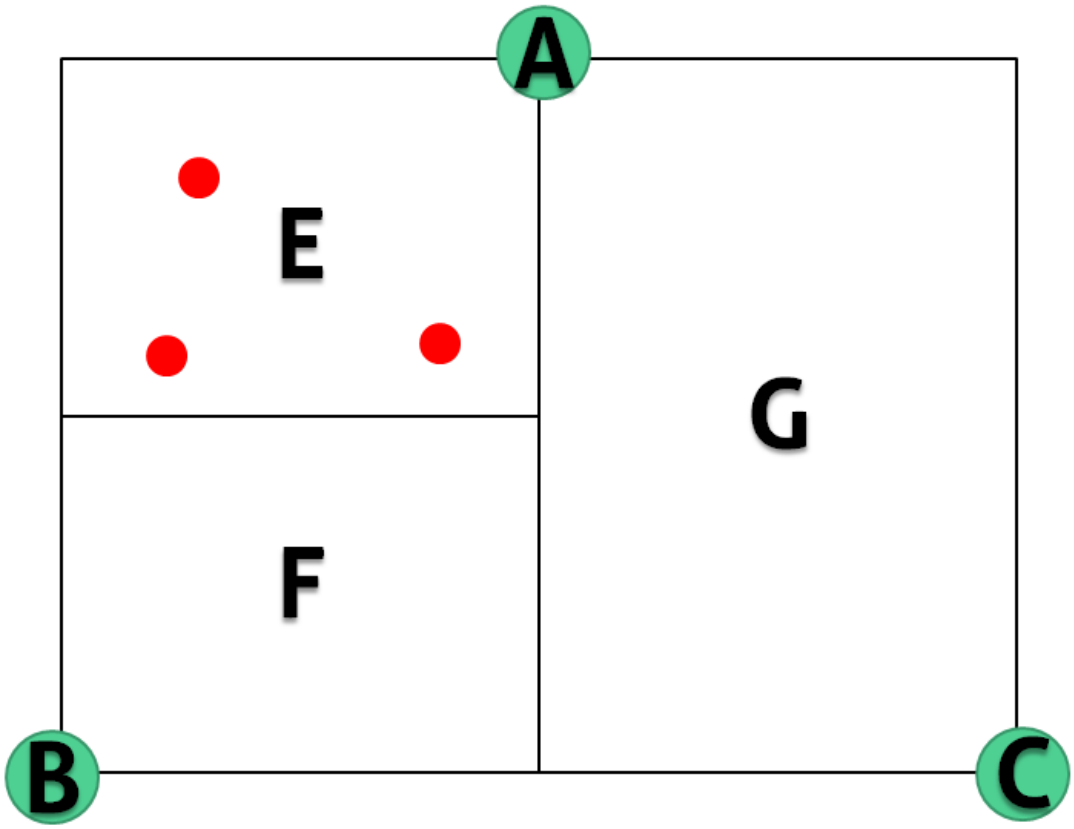


Figure 6 - Calibrating fingerprint E by getting RSSI information from different location in the same room

Figure 7 - when server receives "calibrate" command, fingerprint's RSSI range will be modified based on the RSSI received.

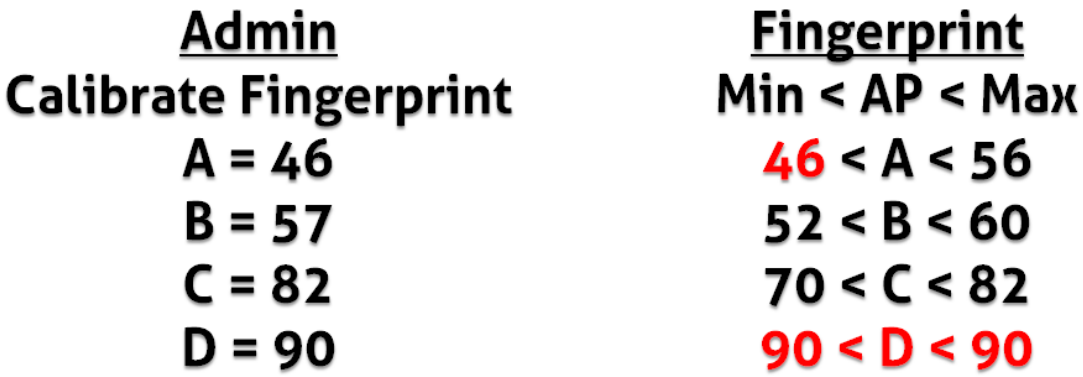


Figure 8 - If a new AP was obtained but the server does not have it's RSSI, the server can include the AP into the table.

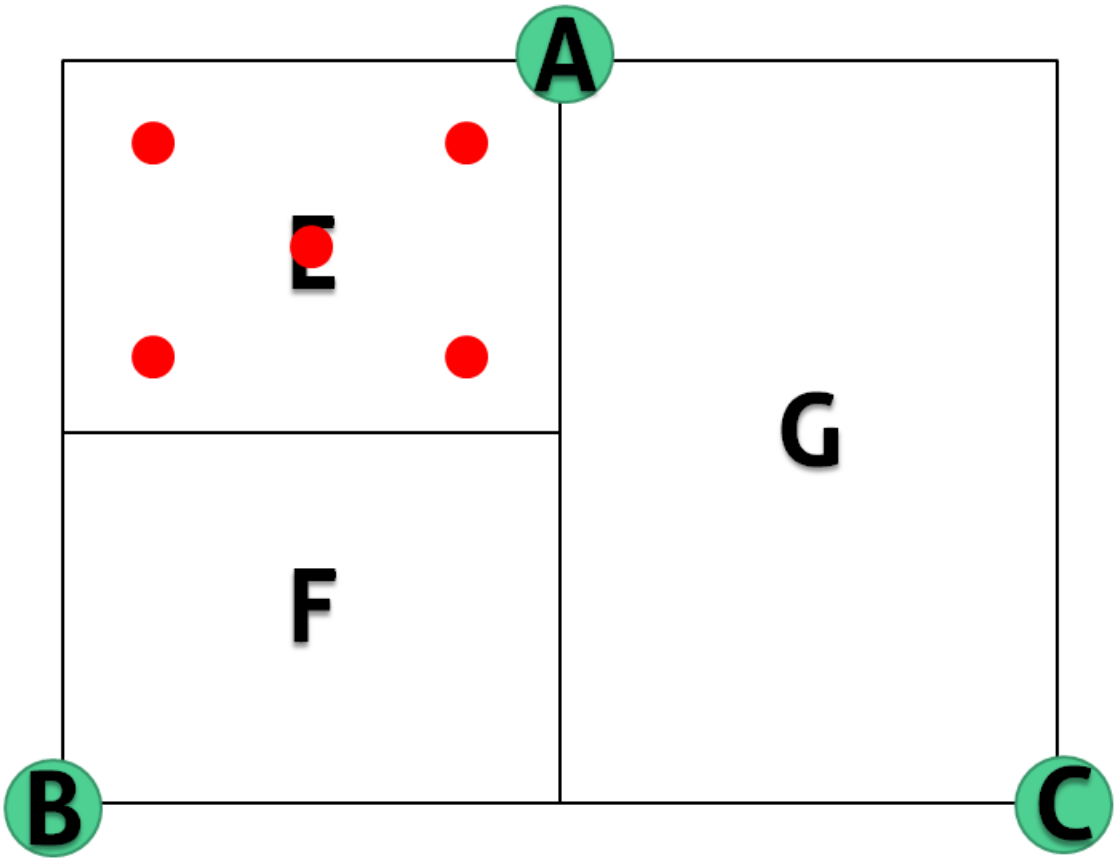
NOTE: each fingerprint can hold up to **20 different RSSI** before the server starts **rejecting new RSSI** for that fingerprint in the calibration process.

NOTE2: different devices provide different levels of RSSI but most of the devices **provide a similar trend line,** a way to normalize the signals is still under testing.

Calibration Tricks

Based on a few tests and experiments, it is found that there are certain ways of calibrating a fingerprint required to obtain room-level accuracy.

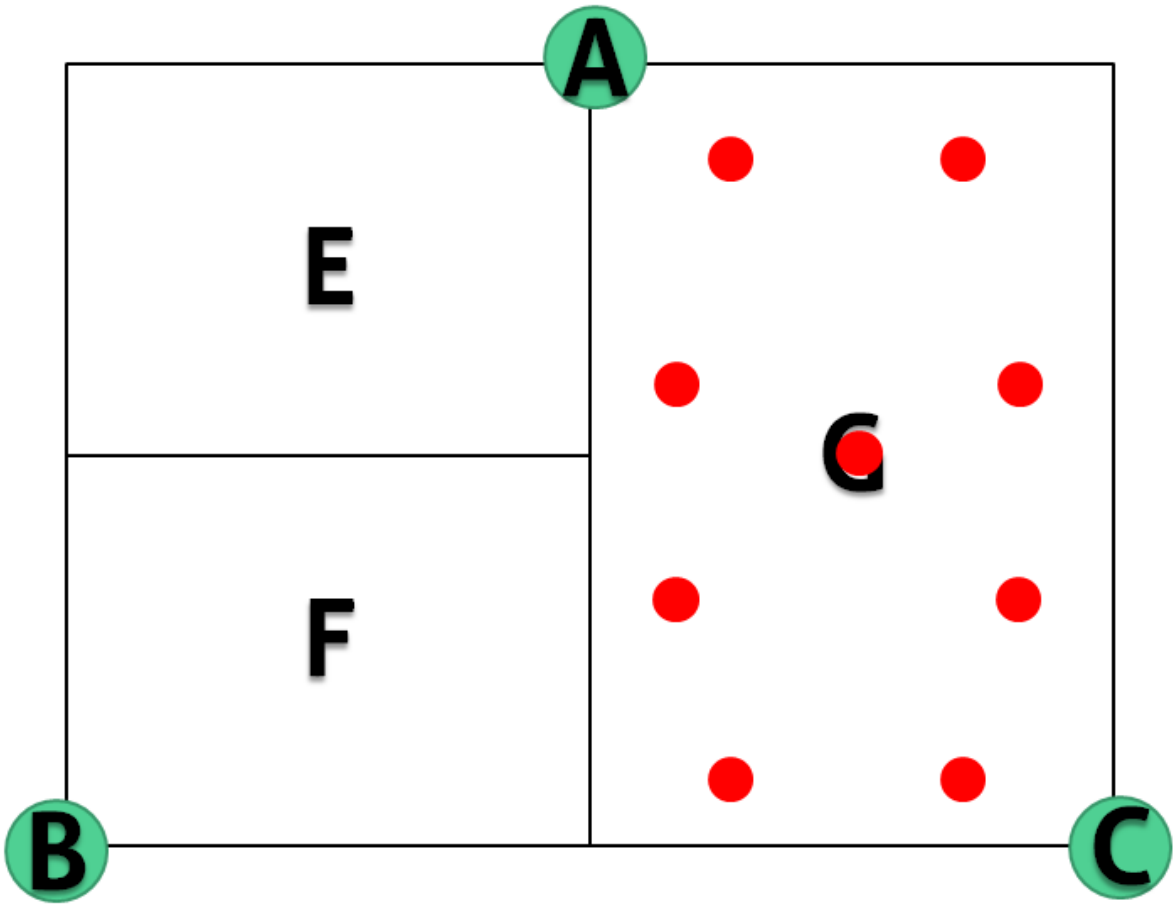
1) Calibrating a medium sized room (requires 10~15 steps to reach from one end to the other end of the room)
    a) Add the fingerprint when you are standing in the middle of the room
    b) standing at the 4 corners of the room facing the middle and then calibrate the fingerprint



2) Calibrating a Large room (Requires 16~20 steps to reach from one end to the other end of the room)

   a) Add a fingerprint when you are standing in the middle of the room

   b) stand at 8 different spots in the room as shown in the figure below to calibrate the fingerprint.



NOTE: DO NOT attempt to calibrate when you're standing near the wall, as it might obtain an **IRREGULAR RSSI** due to how wifi signals are affected when it hits a wall. Based on our testing, this could greatly impact the accuracy of the calibrated fingerprint.

Locating via Data Comparison

After the fingerprint setup and calibrating process, the system will then be ready to compute a device's location. The server will have to **preset the precision level** (3~10) for it to determine **amount of mac address RSSI to pass to determine the right fingerprint for the user's location.**

Note: for example purposes, the signals shown are in positive value. Signals are usually displayed in negative value in dB.

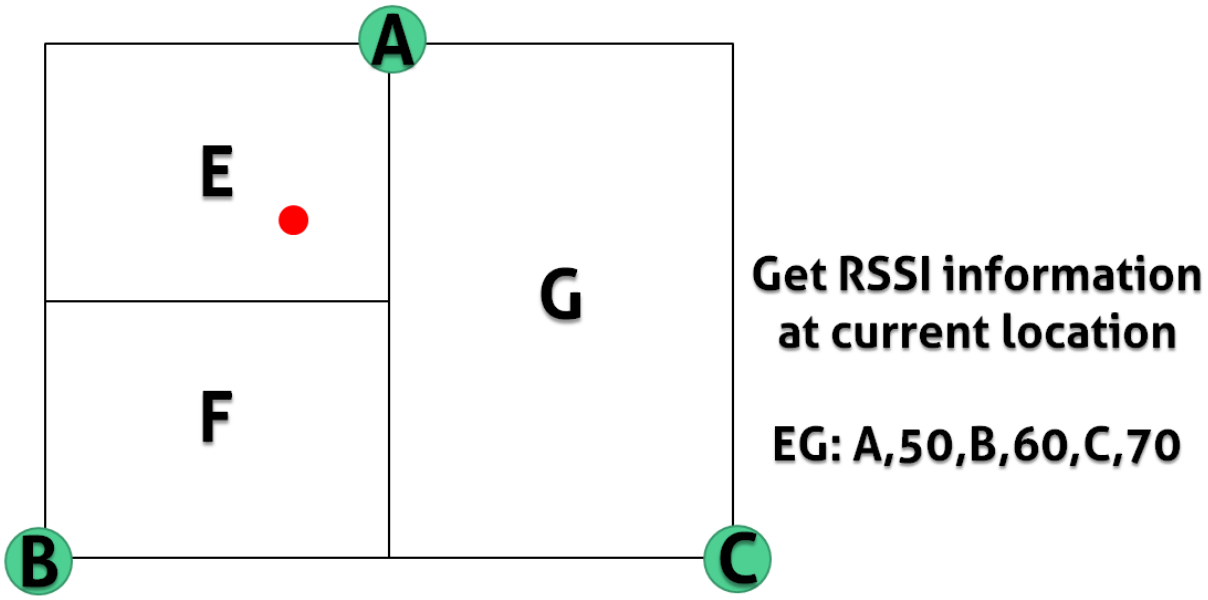


Figure 9 - a user standing at the red circle sends the locate request to server

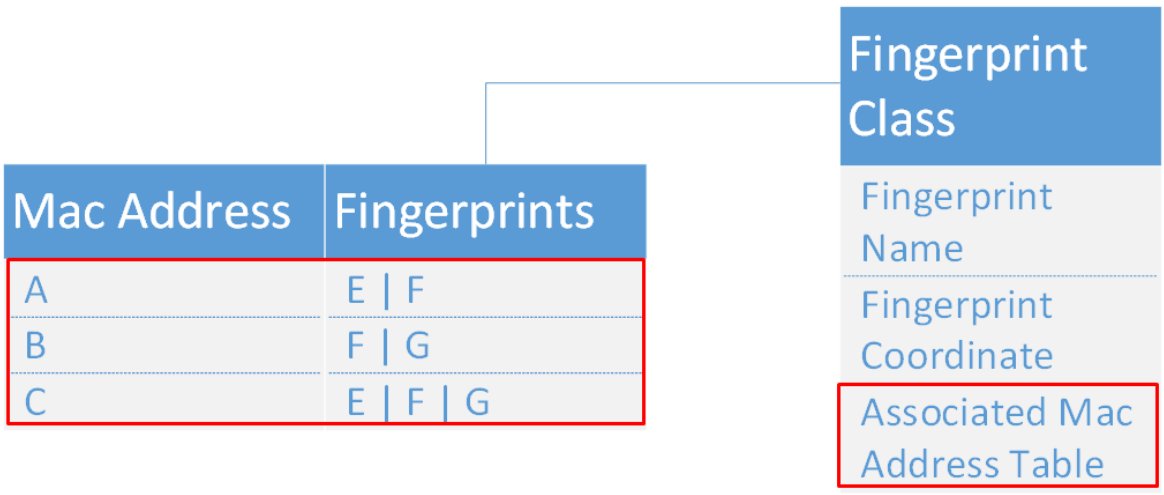


Figure 10 - The server retrieves the fingerprint list to be compared with the person's RSSI based on the Mac Address key provided by the user.

NOTE: right now, the **server will obtain duplicate fingerprints**. HOWEVER, the server **will not check the same fingerprints more than once**. This could be fixed in future release

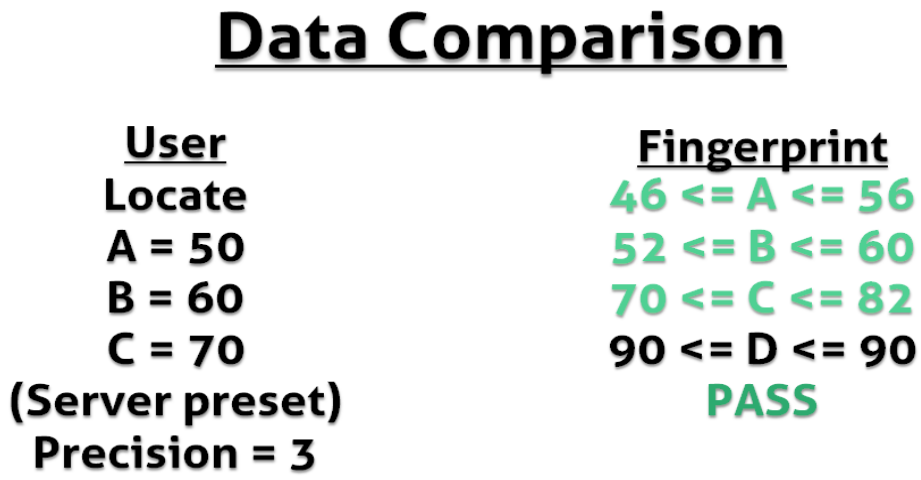


Figure 11 - for precision level 3, only 3 RSSI have to be within range for server to conclude the user's location.

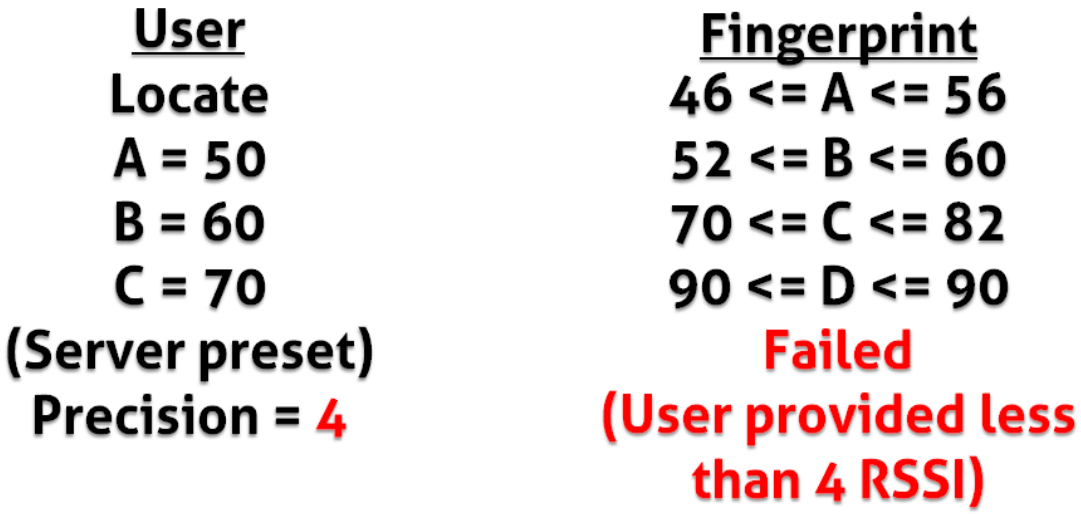


Figure 12 - at precision level of 4, any locate attempt with less than 4 RSSI given will be rejected instantly.

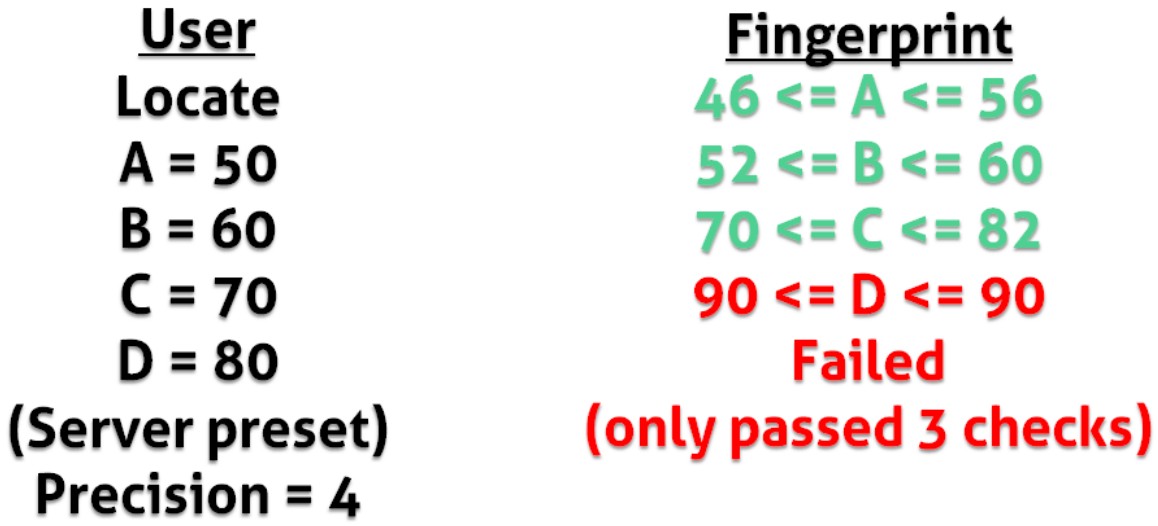


Figure 13 - at precision level 4, 4 RSSI has to be in range to consider the user to in the specific fingerprint.

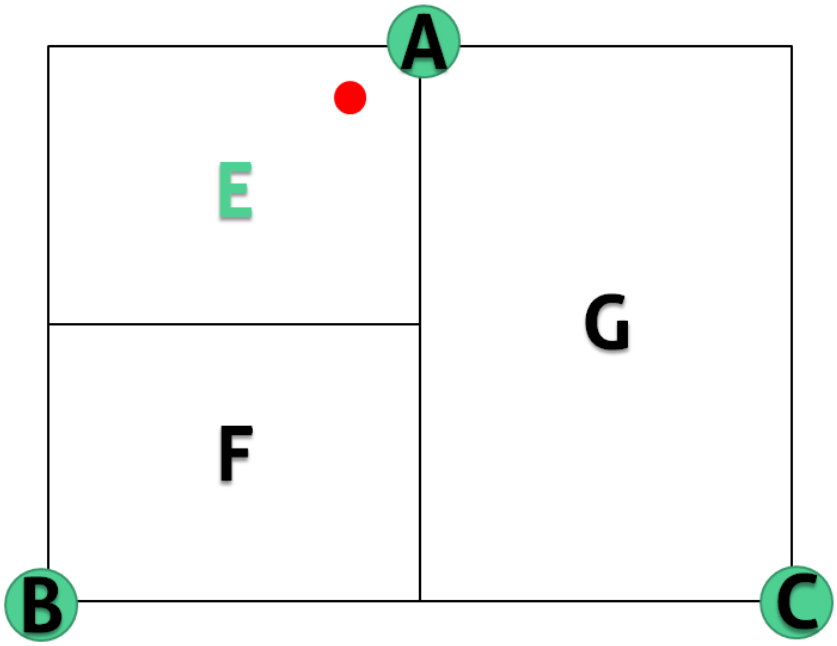


Figure 14 - Server determined the location of the red circle after comparing related fingerprints

Implementation Platform

Server

The server was programmed in C# using Visual Studio 2013 as the IDE. The supported OS for the server to run are **windows 7, windows 8** and **windows XP** (tested using virtual machine) with the only extra requirement to have **.NET Framework 4.5** installed for the server to run without any issues.

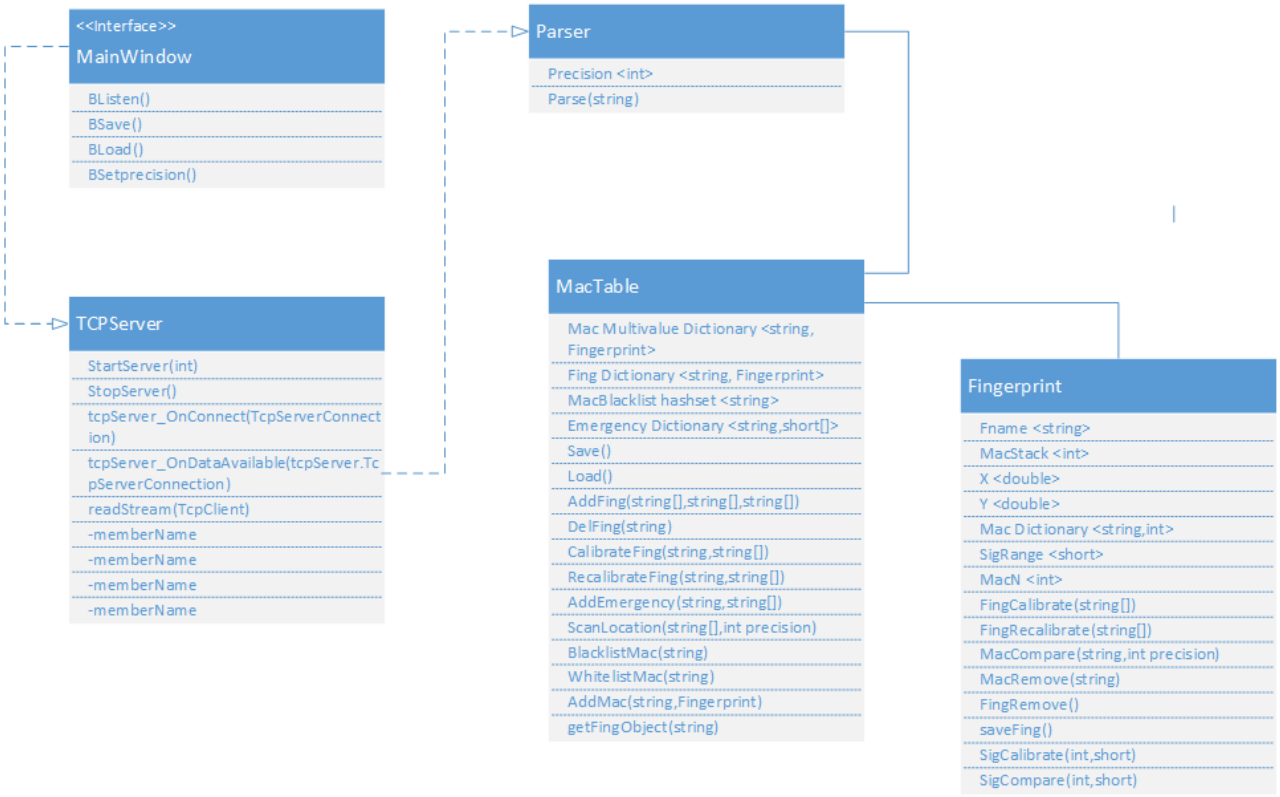


Figure 15 - UML of the server

When the server application starts, the application will display 2 windows, the console window which shows the server's log and the GUI window, which will show the map and the option to modify port, save/load and also precision settings.

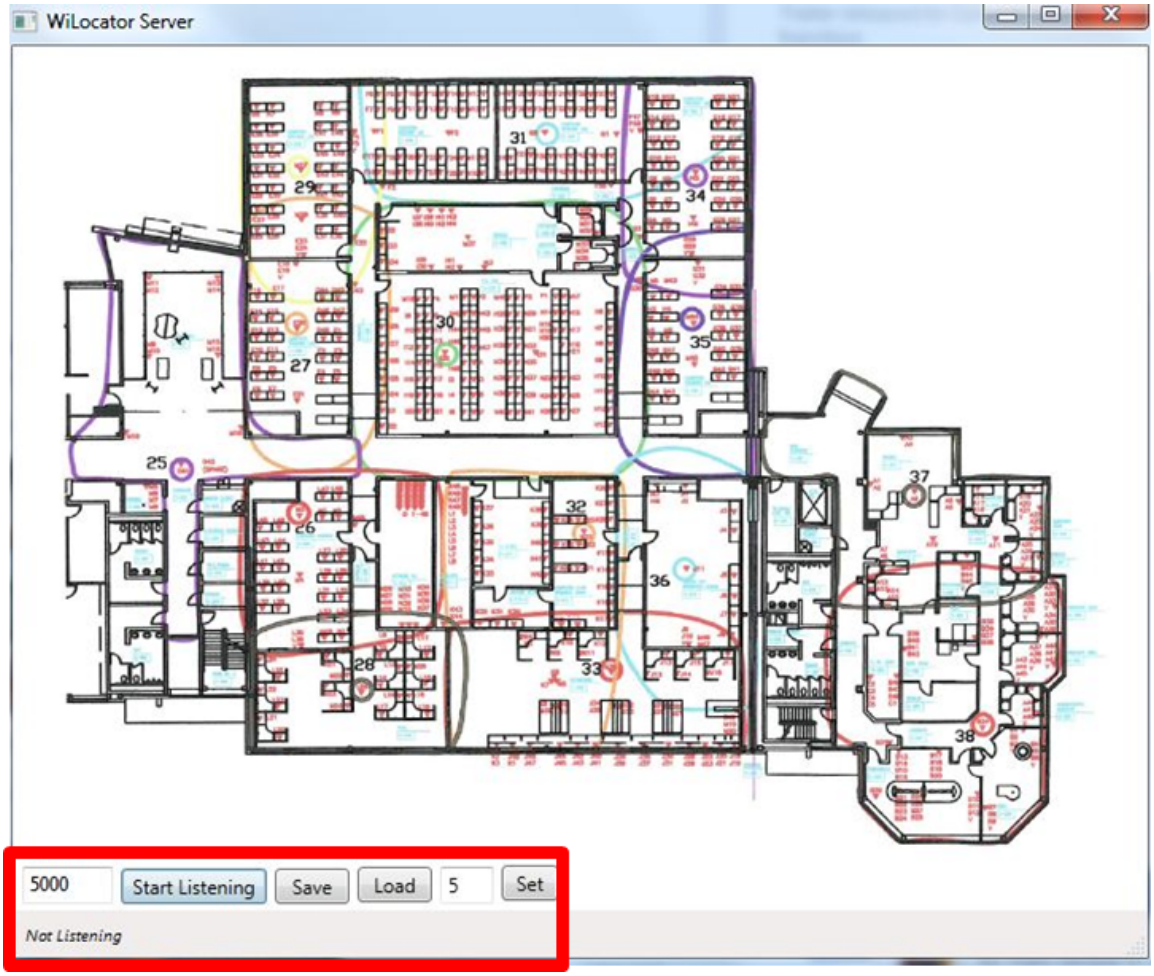


Figure 16 - The GUI of the server with the red box outlining the settings that could be done

Start Listening - on click, will start accepting client at the specific port in the textbox at the of the button

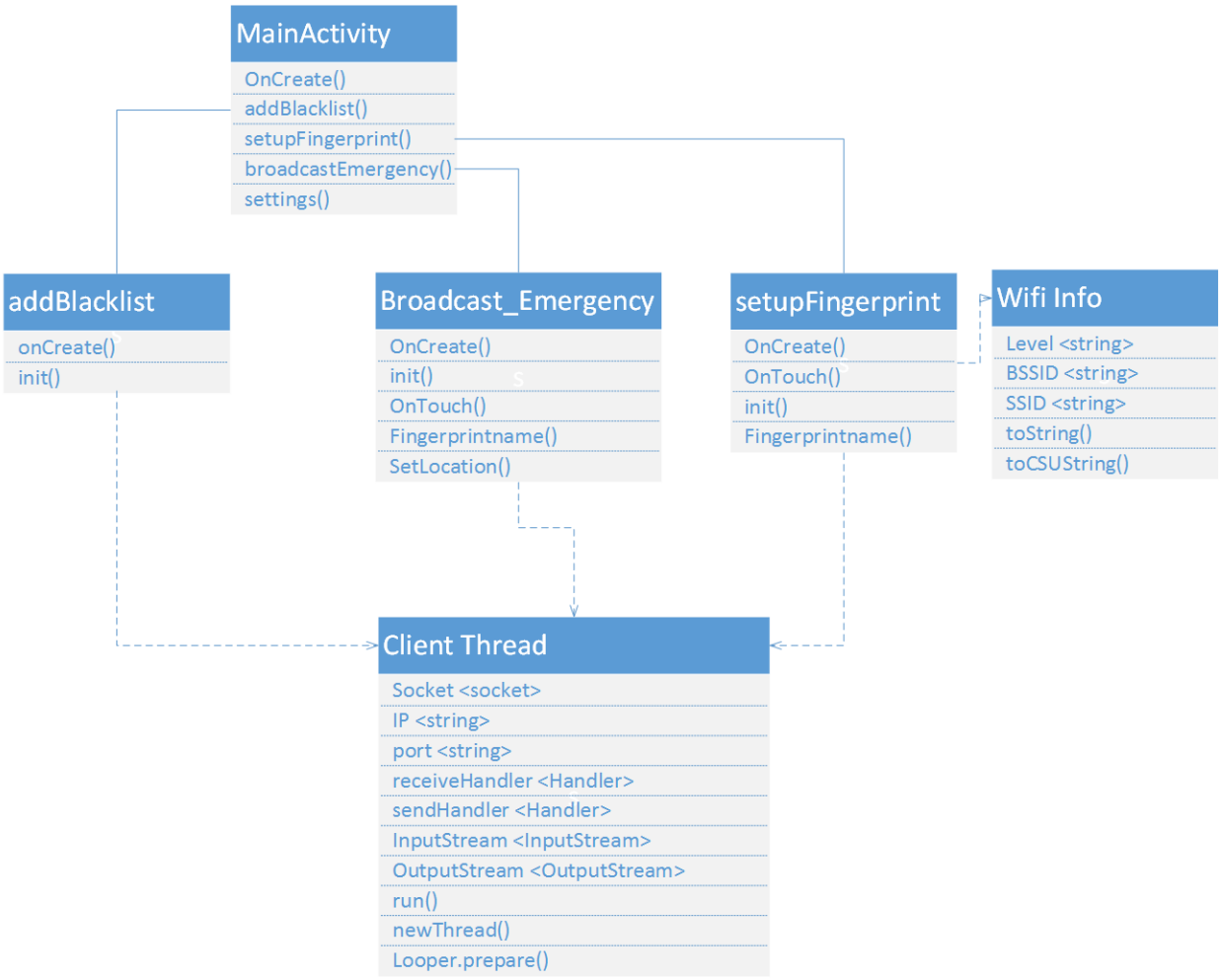
Save - saves fingerprint and blacklist mac address information into a folder named "Backup" (the folder could be found at the same place as the application itself).

Load - loads fingerprint and blacklist mac address information from the “Backup” folder

Set - Sets the locating precision value (3-10), could be changed anytime.

Client

From our design selection, the client code was written for the android platform using Eclipse IDE, an android application was exported and installed on an android phone. The target Android SDK API is set to 20 (KitKat) and the minimum supported SDK API version was set to 11 (HoneyComb). Thus, this client program can run on all android versions from 3.0 to 4.0 and above. Once the client programs is installed and launched on the client device. Below is the view of menu selection of activities that the client can do.



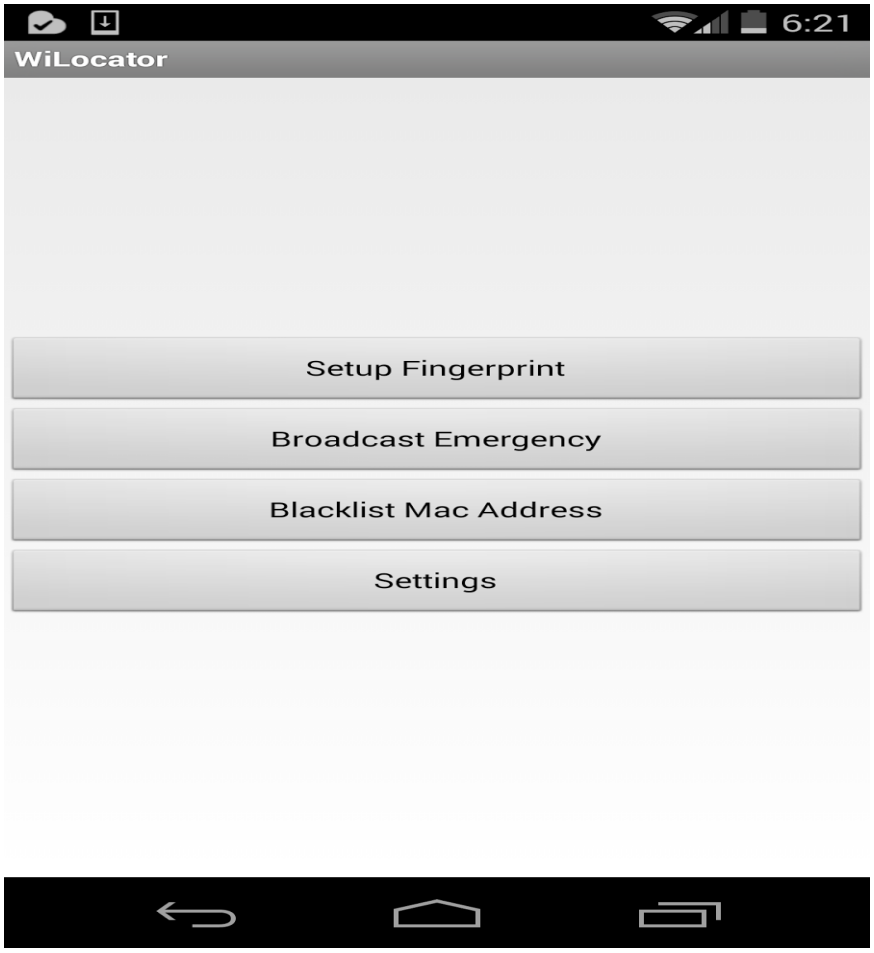


Figure 17 - Main Page

The client can setup fingerprints, broadcast an emergency, blacklist mac addresses, and setup the settings for the server, which include ip address and port number. The most important step is that the client needs to set these settings the first time the admin client launches the application in order to connect to the server that is already running.

Setup Fingerprint

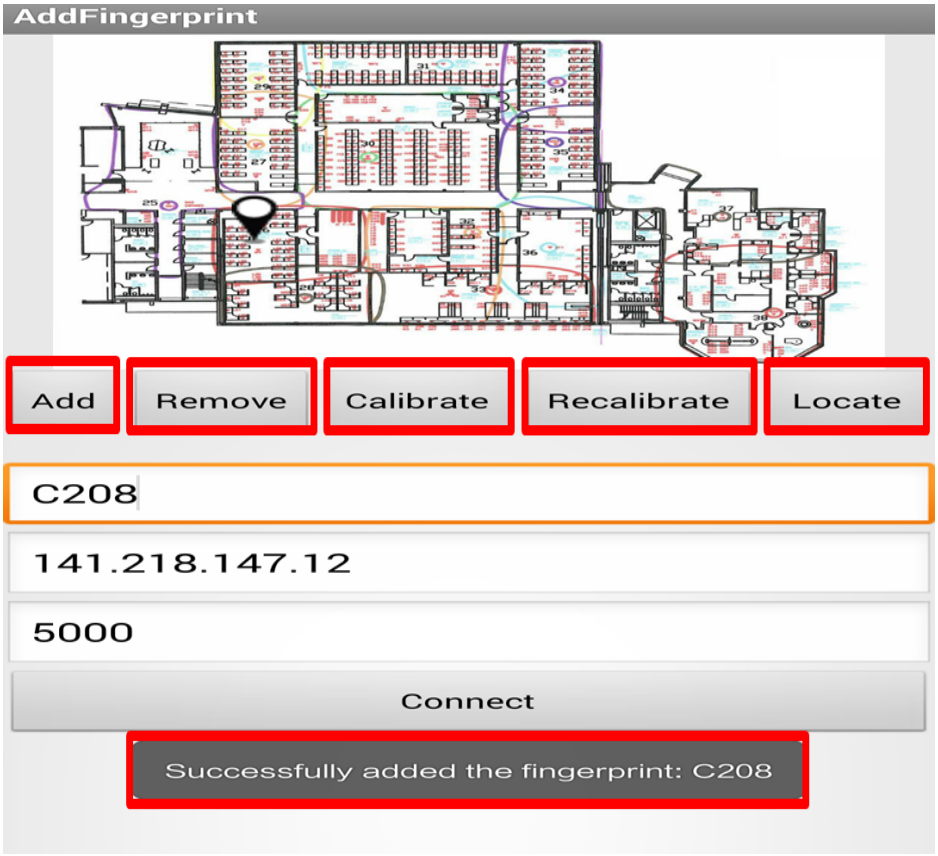


Figure 18 - Setup Fingerprint

In the setup fingerprint screen, the admin client will have to type in the fingerprint name on the text edit box. The admin can touch anywhere on the map to set its position in the building. By pressing the add button, information like fingerprint name, x,y coordinate along with mac address and signal strengths are send to the server, client will receive a reply from the server that the fingerprint had been successfully added. If the previous associated fingerprint had been added, the server will not handle the request. Toast message will appear on the screen stating "[fingerprint name], collision with fingerprint [fingerprint name]"The server can also handle request by removing the fingerprint when admin press the remove button. If no fingerprint were found on the

server, server will reply the fingerprint does not exist.

The calibrate button allows the admin to **calibrate** the fingerprint measurement, in this step, admin will walk around the area in a room and press the calibrate button. This initiates a saved fingerprint with the latest RSSI measurement at that location. The associated measurement of that fingerprint were taken around every corner of the room as well as the center to obtain a more accurate measurement as possible. Every time the calibrate button is pressed, the server will reply the amount of signal strengths calibrated for that fingerprint. On the other hand, if the initiated fingerprint were not found on the server, toast message will receive on the client screen stating that the fingerprint [fingerprint name] does not exist.

Next, the recalibrate button is somewhat similar to the calibrate function but the difference is that this function will remove all the associated fingerprint measurement that had been taken and insert back into the server to compute the client location accurately.

Lastly, after all the setup fingerprint phase was complete, the admin client then can locate themselves manually by pressing the locate button. Since this is the admin side setup, no schedule timer are implemented to locate the admin client automatically. Admin client will then receive a toast message from the server that states the client location.

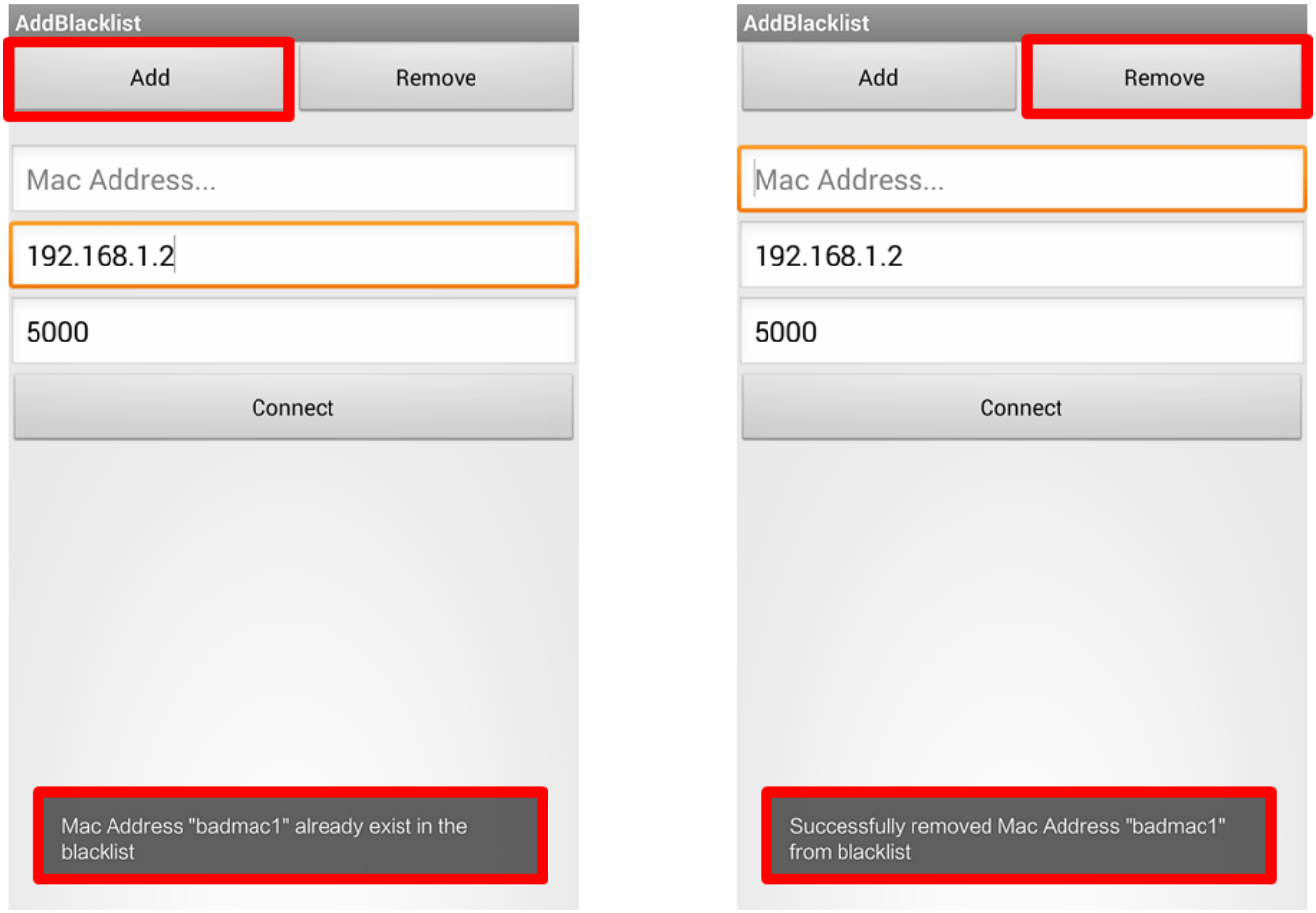


Figure 19 - Black list Mac Address

This is the blacklist mac address screen. As shown on the left figure above, this feature requires user to manually type in the mac address of that access point to add into the blacklist. If the mac address is already added in the blacklist, the client will receive a toast message from the server that the mac address notifying the client that the mac address is already existing. Client can also remove the mac address from the blacklist so that particular access point can be used again during calibration phase.

Broadcast Emergency Wifi Info

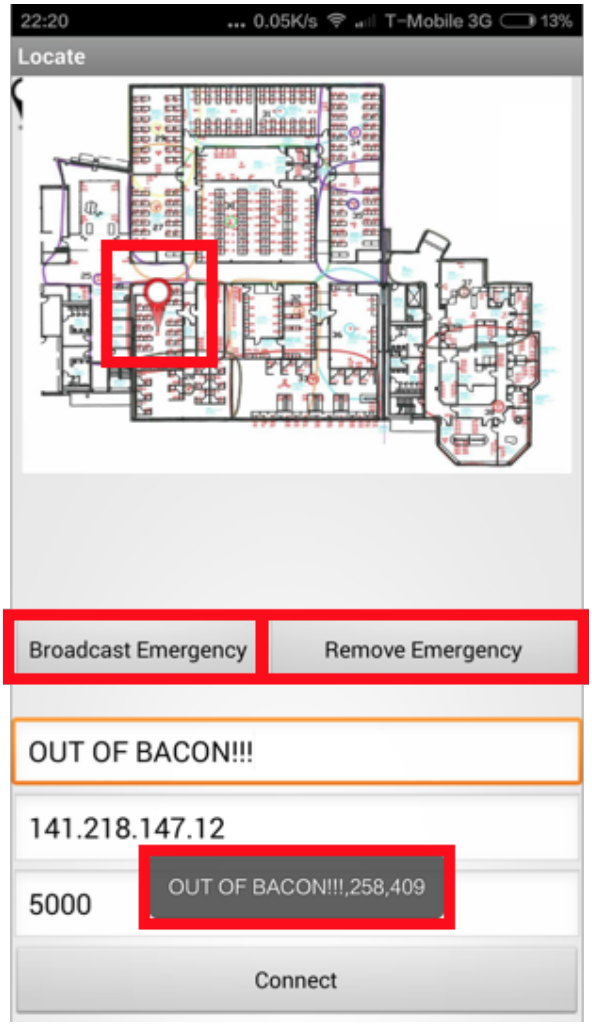


Figure 20 - Broadcast Emergency

In this broadcast emergency page, the client can press anywhere on the map canvas and a red pin will be shown on the screen thus obtaining the x,y coordinate of that emergency. In the text edit box shown on the figure above, client will have to specify the emergency name. The server will be notified when the broadcast emergency button is triggered. The server will broadcast to connected clients stating the emergency name and x,y coordinates of that emergency. The server will stop broadcasting the emergency message when the admin press the remove emergency button.

Performance

Initial Plan to Measure Performance

In order to achieve our goal for this project, a plan was devised. This plan consists of several procedures that measure the performance and reliability of the working project. The measurement of this test was being able to record a stable RSSI during the calibration phase. We managed to prove that the user can locate themselves indoors up to a room level accuracy and also notify the server an emergency with easy setup on the client side. Similarly, this test also involving the server capability to broadcast emergency message to all client that are connected.

Methods

The setup fingerprint phase which include adding, calibrating, and locating fingerprint were all designed in one view. This layout and functionality followed our devised plan that allowed the admin client to do the setup and testing all in one page. As mentioned in the plan above, between three to fifteen access points with the strongest signal strength are scan in achieving a more stable measurement during the calibration phase. In order to calibrate the associated fingerprint accurately, the admin client will have to walk the entire perimeter of the space including the center. This process was repeated in every space until a constant RSSI values are saved on the server. Since the blacklist mac address function was implemented, this function can avoid the use of specified malfunctioning access points during calibration. Lastly, the accuracy of our location increases as more calibration of each fingerprint is taken. There is no supporting data to show this test locating performance other than the observations of the program itself.

Testing

Clutter Test

We tested this application against clutter and our goal is to achieve up to a room level accuracy. Clutter is a collection of objects lying around in a room that may cause signal interference. Example of clutter are physical barriers like walls, floors, and metal objects. Although Wi-Fi signal waves are invisible to the naked eye, they can be affected by this clutter. Clutter like metal absorbs and blocks Wi-Fi signal making it difficult to record uniform Wi-Fi information. Furthermore, these access point signals can also be affected by other appliances in the building that use the same frequency bands as the access points. These include video cameras, television cable box, video cameras, and even other wireless routers in that area.

Our test procedure in testing this project consists of only three steps, setup fingerprints, calibrate fingerprints, and test locating. Since we tested this project against clutter, we managed to achieve up to a room level accuracy with a precision value between five to seven in determining the user's location.

Problems Encountered and Possible Fix

Problem 1: DeadZone

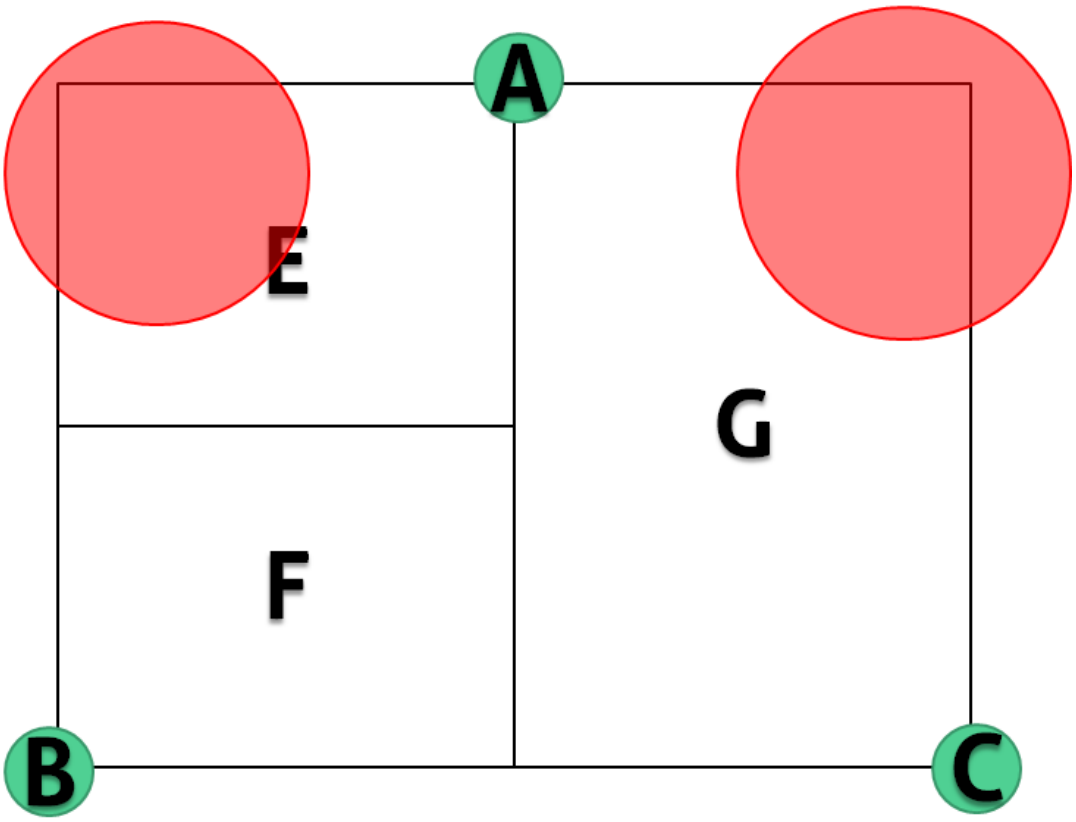


Figure 21 - Dead Zone image

The figure shown above is known as a “DEAD ZONE”. When the user walks up to these red spots and tries to locate themselves, the server will not be able to compute the users location because the access points in range in that area are less than three. In order to localize a user’s location in this red spot, a possible fix to this problem is to have beacons hung up on the corner of the wall. Whenever the user walks up to the “DEAD ZONE”, the client will scan that beacons signal strength information and send it to the server to compute the client location.

Problem 2: Ambiguous Zone

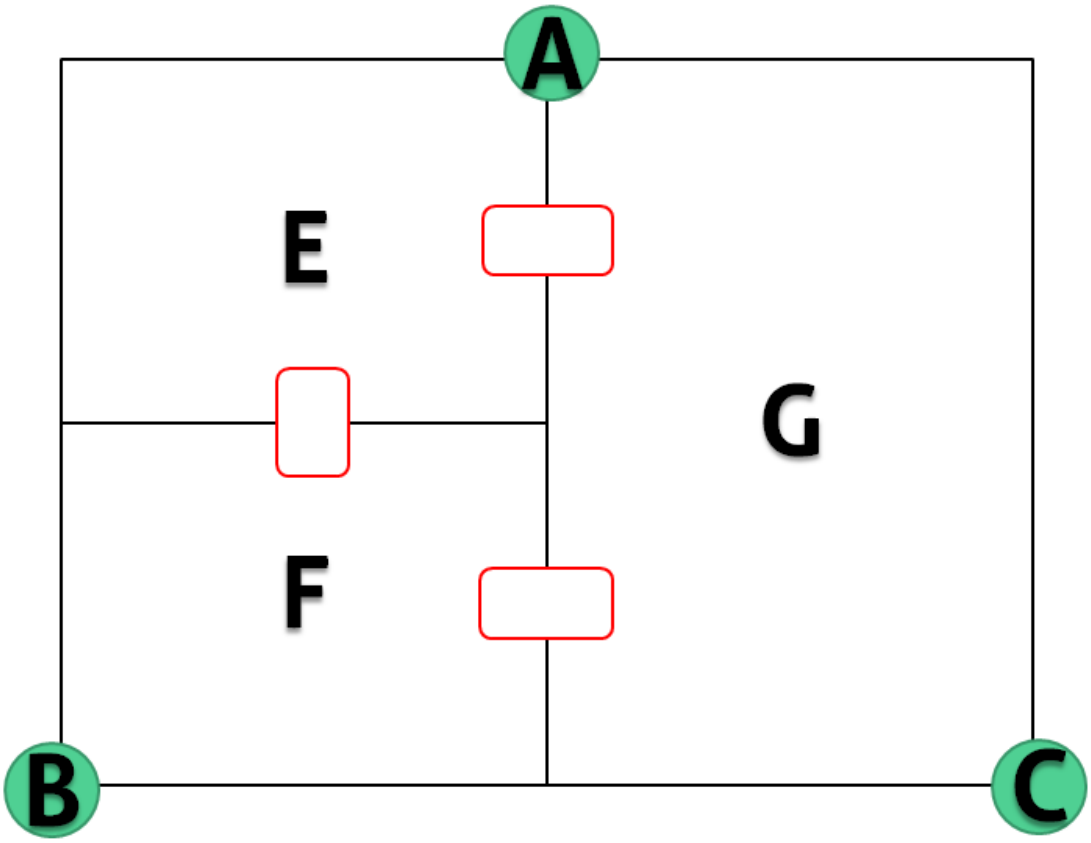


Figure 22 - Ambiguous Zone image

This figure shows the ambiguous zone that occurs on the doorway. When the user stand in between location E and location F, the server could not determine which fingerprints that was added and calibrated. There is no fix for this project for now, as this problem is expected during the testing period.

Problem 3: Tested Area Against Clutter

We tested this application in two different places, first in our apartment and second at Parkview Campus. Since we needed at least fifteen access point to test our development, testing this application in Parkview campus is not a factor in determining the user location since there are more than fifty access point placed across every building and room.

During the calibration testing in Parkview campus, we noticed that a more constant RSSI information was recorded due to less clutter in the room. Since most of these access points in Parkview are placed on the ceilings, objects like chair and table may not be considered as clutter that causes signal interference.

However on the other hand, when tested this project in our apartment, we noticed inconsistent RSSI information recorded. For example in our case, our router are placed under the desk table. This desk table are objects that may be considered as clutter causing irregular Wi-Fi signal strength. Appliances around the house takes into account too that may be considered as factors affecting our calibration testing, thus resulting in no optimum precision value can be achieve due to irregular signal strength in the area.

Problem 4: Tested Models

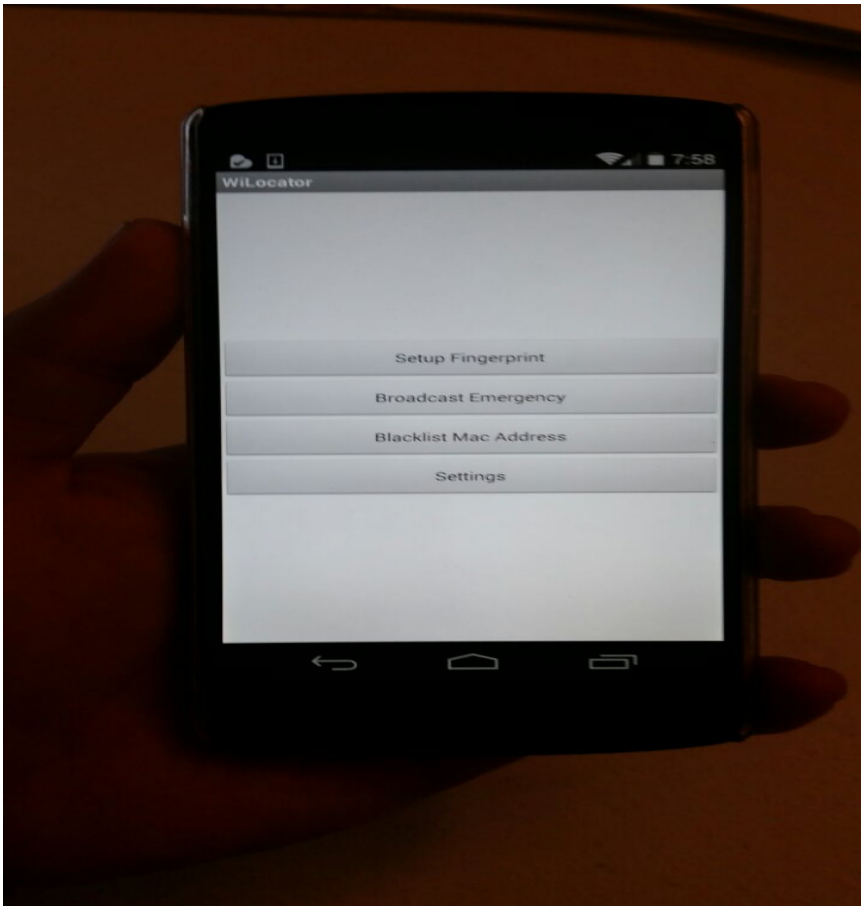
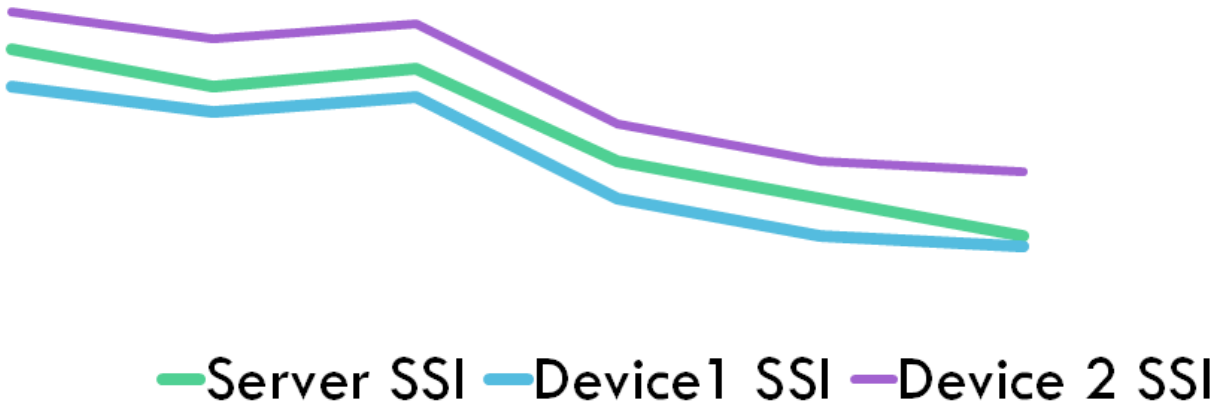
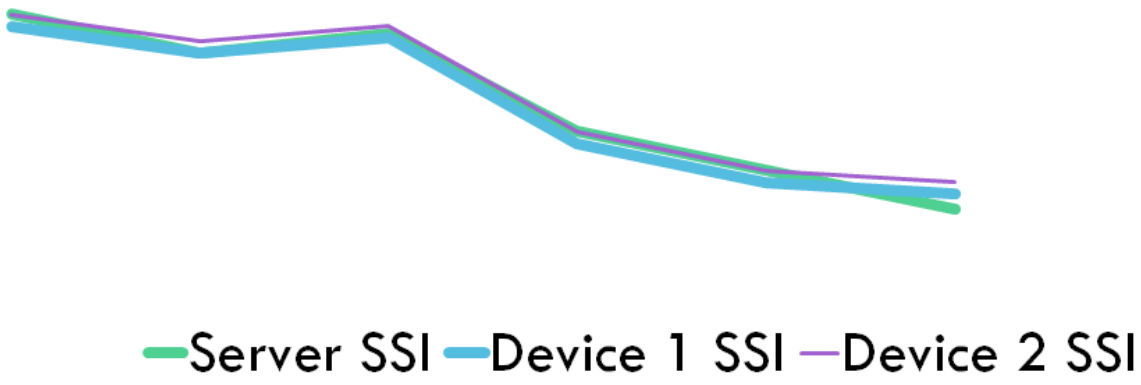


Figure 23 - Nexus 5 were use for testing

We tested this project on different type of android platform, be it a phone or a tablet. Example of tested model was Nexus 5 and the Xiaomi 4. We noticed the difference between this devices when scanning and recording the received signal strength. This is because different android platform provide different types of software and hardware, thus making it difficult to obtain a uniform RSSI for the server to compute precisely. We can conclude that different devices have different RSSI values.



In our future release, a possible fix to the mentioned problem is when different devices send the RSSI information during calibration mode to the server, the server then compute and compare the device's average RSSI with the server RSSI and then normalize the RSSI information of the device's average RSSI with the server RSSI.



Maintenance

Client

Since the client source code is uses the API Level 20 which can run on almost all android version, there are no need for maintenance for now. In the future if you want to use this project, you can modify the manifest file in the android source folder and set the API level accordingly to your specification. Some material design features like the material theme and custom activity transitions are only available on Android 4.0 (API level 20) and above. However, you can design your apps to make use of these features when running on devices that support material design and still be compatible with devices running previous releases of Android. You can refer to the android API package which can be found online at http://developer.android.com/reference/packages.html. The oracle java library can be retrieved from http://docs.oracle.com/javase/7/docs/api/.

Server

If you are to implement your own client to be used with our server or modify our server, it is recommended that you follow these guidelines:

- **Format the string**s to the appropriate format (read **"IPS EXAMPLE INPUT.txt**")
- **All mac table function call should be going through the Parser()** method to maintain program flow
- Any method that **updates any dictionary** should be done **inside a "Lock(object)" block** with the respective object that's gonna be updated
- If multiple objects are being updated at the same time, make sure to **lock object in the same pattern to prevent race condition or program freeze.**
- **Do not call the fingerprint class's method outside of mac table class**, you will hate yourself in the future when attempting to debug the program.
- **Document type by entering 3 slashes ("///") before a type** and visual studio will automatically produce an XML format documentation (Guide)
- **Modify ScanLocation()** function if you wanna change the **locating algorithm**
- **Confirm function works before allowing parser to call the functions.**
- **TcpServerConnection** and **tcpServer.TcpServerConnection** are **different** Visit the following link for tcpServer class information (Link)
- **Alchemy class is not used currently,** it is used for websocket purposes only (if you want to implement a browser as client, use alchemy class instead of tcpServer class)

Security

Since this project is a proof of concept, there are no security that has been implemented in this project.

Future Security Implementation

Since our project is a proof of concept, no security has been implemented. In our future release, a secure login system can be implemented to grant access to admins only.
The login system implemented will defend several possible attacks like SQL injections, network eavesdropping and brute force attacks.

In addition, the communications between the client and server will be encrypted to protect client privacy information. SSL can be created and used to ensure a secure connection between both client and server.

Future release ideas

There are many ideas that come to mind when one thinks of indoor positioning and the use of our system. These are some we think are within the scope of this project and would benefit the project.

The next release

- **Normalizing RSSI from different device (important)**
- A better formula for calculating a device's location.
- Converting from embedded to external database to support buildings with more than 1000 AP
- Design improvement of client and admin apps (displaying device location on GUI)
- Login credentials for admin
- Using SSL to encrypt connections
- GUI to show where user are instead of displaying text

Future releases

- Letting client app send emergency to server (only admin can do so currently)
- Add a tracking feature (storing locating logs on server side)
- Feature to help admins setup AP and locate dead zones

Deliverable

1. The Android Application Package (Client Program)

Client program that was developed in android and export to an APK. The client can established the connection between the client and server. Information were send over a TCP/IP socket connection to the server to determine the user location in an indoor area using Wi-Fi.

2. The Server (Executable server program)

A functional server that was developed for windows 7 compiled into a folder containing all necessary libraries for the executable to run are included.

3. Software Documentation

The client codes was documented created using JAutodoc plugin for Eclipse IDE. The documentation contains API's that defines the class and methods used for future reference. The server's documentation was done using Visual Studio 2013 Built-in XML documentation which visual studio will automatically produce when compiling the code.

4. Project Report

This project report was written by three members Kelvin Yap, Davion Teh and Rodney Dulin and delivered to Dr. John Kapenga at the end of the Fall 2014 semester, CS4910 - Senior Design II.

5. Presentation

The presentation on our senior design project was conducted on December 2nd 2014 at 11 A.M. The presentation consists of our design decision, implementation, problems encountered and screenshot of our working project.

Legal

Licenses

This project is free software; it can be redistribute and/or modify it under the terms of the GNU General Public License Version 2 as published by the Free Software Foundation. This work is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose.

Intellectual Property (IP)

This project is being developed as a Senior Design project for WMU, Western Michigan University, under the supervision of Dr. John Kapenga. WMU will retain the intellectual rights to the software.

Non-Disclosure Agreement (NDA)

No non-disclosure agreement is being used at this time. The project is maintained on GitHub, which is freely and openly accessible to anyone who wishes to view it, and is thus tracked by search engines such as Google, where it is able to be searched for by anyone on the planet.

Glossary

GPL 2 - GNU General Public License version 2

CEAS - College of Engineering and Applied Sciences at Western Michigan University

GUI - Graphical User Interface

RSSI - Received signal strength indication

RSS - Received Signal Strength

API - Application programming interface

ADT - Android Development Tools

APK - Android Application Package

SSL - Secure Sockets Layer

Reference

tcpServer - http://www.codeproject.com/Articles/488668/Csharp-TCP-Server

tcpClient - http://stackoverflow.com/questions/20263522/android-tcp-server-client-communication

Alchemy Websocket - https://github.com/Olivine-Labs/Alchemy-Websockets

Microsoft experimental collections -

https://www.nuget.org/packages/Microsoft.Experimental.Collections

Multi Value Dictionary -

http://blogs.msdn.com/b/dotnet/archive/2014/08/05/multidictionary-becomes-multivaluedictionary.aspx

Dictionary - http://msdn.microsoft.com/en-us/library/xfhwa508%28v=vs.110%29.aspx

License - http://www.gnu.org/licenses/licenses.html

Redpin - http://www.redpin.org

ADT - http://developer.android.com/tools/sdk/eclipse-adt.html

Android Programming - http://www.tutorialspoint.com/android/

Position Tracking Using WiFi - http://scholarworks.wmich.edu/cgi/viewcontent.cgi?article=3411&context=honors_theses

Redpin info - http://www.vs.inf.ethz.ch/publ/papers/bolligph-redpin2008.pdf