An Android Application to measure Wi-Fi Signal Strength Campus-Wide (Integrated with Google Maps API)

KARTHI KISHORE JAKKLI SOUNDER, University of Nebraska – Lincoln

With the advent of smart phones, mobile computing has been revolutionized and many new applications have been introduced especially in the world of wireless networking. In this project, we propose to develop an android application that identifies the extent of Wi-Fi network coverage across UNL campus by taking into account the signal information. The coverage map reports the signal strength and in addition, it is also a profiling application which gathers the information related to the usage of system resources like battery, network and other sensing resources/information. We believe that this data may help to visualize the locations that have the greatest coverage, medium coverage and low coverage across campus, to identify a behavior pattern which in turn helps to analyze the network related information.

Categories and Subject Descriptors: **C.2.1 [Network Architecture and Design]: Wireless Communication**

General Terms: Wi-Fi, Android.

Additional Key Words and Phrases: **ACM Reference Format**: Gang Zhou, Yafeng Wu, Ting Yan, Tian He, Chengdu Huang, John A. Stankovic, and Tarek F. Abdelzaher, 2010. A multi-frequency MAC specially designed for wireless sensor network applications. *ACM Trans. Embedd. Comput. Syst.* 9, 4, Article 39 (March 2010), 6 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Android is a software stack for mobile device that includes an operating system, middleware and applications. Android, powered by Linux kernel allows developers to write managed code in java language, controlling the device via Google developed java library. Not like other famous rivals such as Microsoft window mobile or Symbian OS, android use developed java library as it's a complete dynamic platform offering powerful support for embedded devices that must maintain some form of dynamic behavior. Moreover, java runtime environment can be integrated into almost any embedded device while java virtual machine includes interfaces that allow it to be readily integrated with RTOS and other native library. The RTOS supports multi-thread (scheduling), memory management, networking, and peripheral management for java VM.

The performance of a Wi-Fi wireless network connection depends on signal strength. Between a physical device and a wireless access point (WAP), the signal strength in each direction determines the data rate available on the corresponding connection. Either of these methods namely wireless adapter utilities, operating system utilities and Wi-Fi locater devices can be used to determine the signal strength of the Wi-Fi connection.

The main motivation for the project comes from a lack of an application that tracks network strength in mobile devices. The Wi-Fi signal is sometimes hard to penetrate farther at distant locations like bus stops, car parking lots, campus loops and recreation centers etc. on campus.

To identify the signal strength of Wi-Fi across locations campus wide and thus the extent of coverage, we have found ways of helping to automate this process of searching weak access points with the help of an android application. Our application focusses to identity the Wi-Fi network coverage across various locations campus wide. An interesting feature of this application is the mapping of locations with the google maps API based on the amount of coverage available. It also serves as a profiling application which gathers the information related to the usage of system resources like battery, network and other sensing resources/information.

2. ANDROID ARCHITECTURE

Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown below in the architecture diagram.

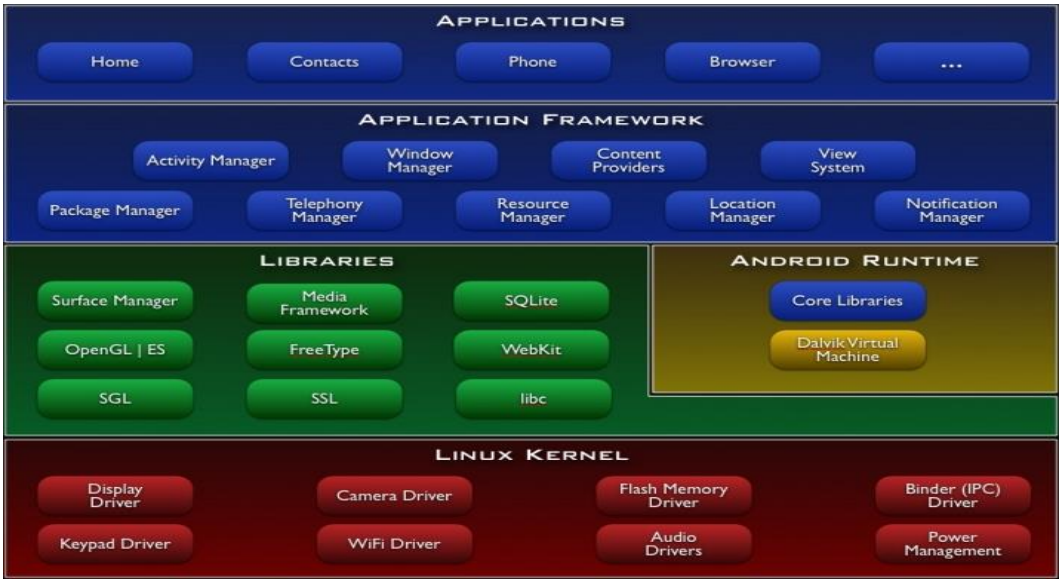

Figure 1 : Android Architecture

- **Linux kernel**

At the bottom of the layers is Linux - Linux 2.6 with approximately 115 patches. This provides basic system functionality like process management, memory management, device management like camera, keypad, display etc.

- **Libraries**

On top of Linux kernel there is a set of libraries including open-source Web browser engine Web Kit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

- **Android Runtime**

A key component called **Dalvik Virtual Machine** is a kind of java virtual machine specially designed and optimized for Android. The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine. The

Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

- **Application Framework**

The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

- **Applications**

All the Android applications are listed at the top layer. Examples of such applications are Contacts Books, Browser, and Games etc.

3. COMPONENT

The interface for the application is optimized for single/double screen phones based on a normal activity view that allows displaying only interesting data on the main part of a screen and allows switching between different kinds of information. An example of the interface is represented on figure below. The interface is in a single screen landscape mode for maps functionality and portrait mode for others.

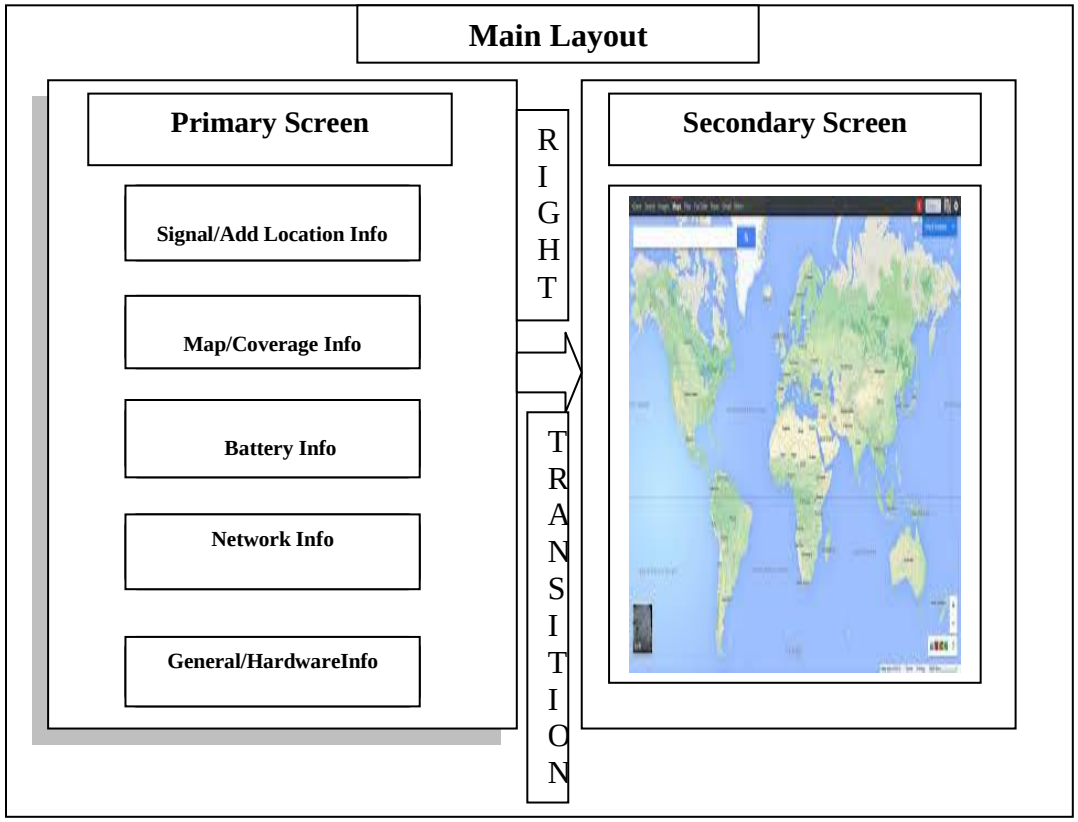

Figure 2 : Layout Structure

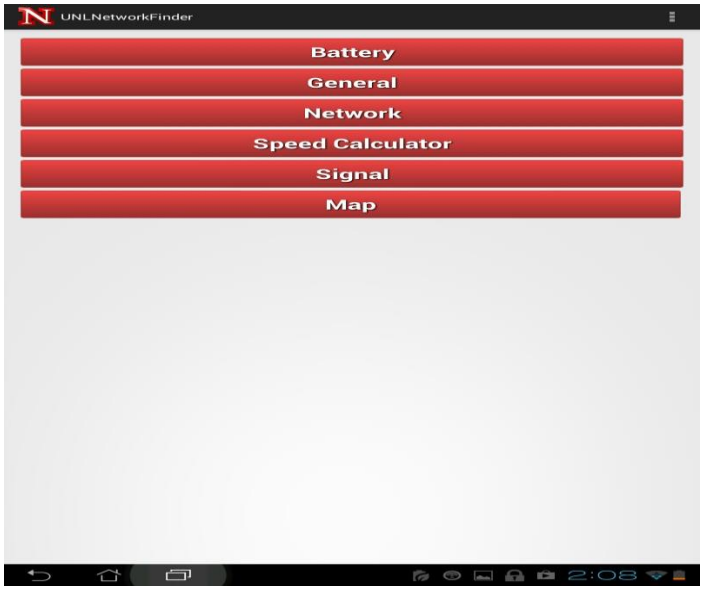

Figure 3: Main Menu, Portrait Mode

The menu bar is presented in Figure 3. It consists of a relative layout oriented horizontally with five buttons. Each button controls either displaying the content or some kind of action. When a button click event is initialized, new activity slides from left to right using animation/translation. The functions of buttons are described as below:

- **Battery** : Displays the battery related information. (Level, Plugged, Status, Temperature)
- **General** : Displays the hardware related information.
- **Network** : Displays the network specific related information (MAC, IP Address, DNS Server)
- **Speed Calculator** : Performs speed test
- **Signal** : Menu to add new location.
- **Map** : Google Maps that displays coverage information using different markers based on signal strength.

The system resource information functionalities like battery, hardware and network uses Table layout for display purpose.

4. FUNCTIONALITIES

4.1 Battery

The battery battery-life impact of performing application updates depends on the battery level and charging state of the device. The impact of performing updates while the device is charging over AC is negligible, so in most cases maximizing the refresh rate whenever the device is connected to a wall charger. Conversely, if the device is discharging, reducing your update rate helps prolong the battery life.

The ***BatteryManager*** broadcasts all battery and charging details in a sticky Intent that includes the charging status. Because it's a sticky intent, simply calling registerReceiver passing in null as the receiver, the current battery status intent is returned

```
this.registerReceiver(this.batteryInfoReceiver, new
IntentFilter(Intent.ACTION_BATTERY_CHANGED));
```

The charging status can change as easily as a device can be plugged in, so it's important to monitor the charging state for changes and alter your refresh rate accordingly. The BatteryManager broadcasts an action whenever the device is connected or disconnected from power.

```
int plugged= intent.getIntExtra(BatteryManager.EXTRA_PLUGGED,0);
int status= intent.getIntExtra(BatteryManager.EXTRA_STATUS,0);
```

The current battery charge can be found by extracting the current battery level as shown below.

```
int level= intent.getIntExtra(BatteryManager.EXTRA_LEVEL,0);
```

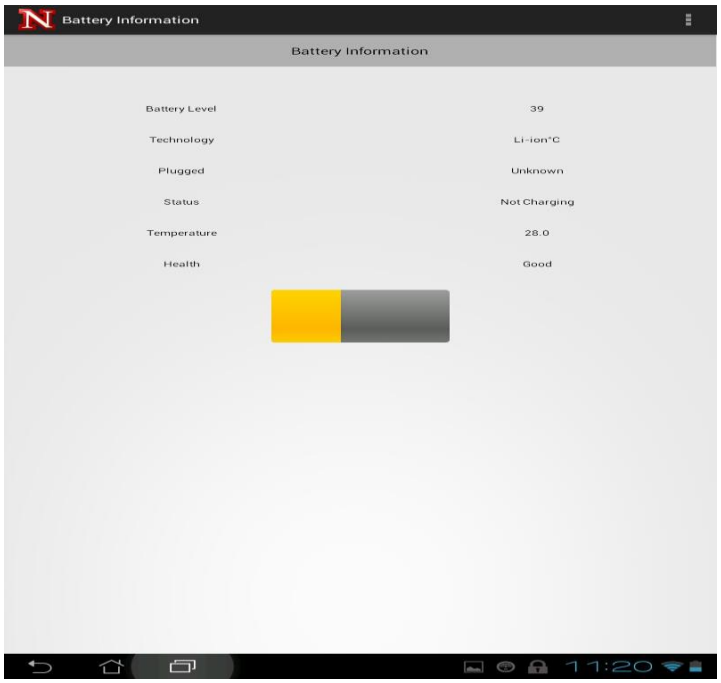

Figure 4: Battery Information

4.2 General / Hardware

The package ***android.os.build*** renders the information about build, extracted from system properties.

```
//Get Hardware
String hardware = android.os.build.HARDWARE;
```

```
                //Get Device
        String device = android.os.build.DEVICE;
                //Get Model
         String model = android.os.build.MODEL;
                //Get Device OS
        String model = android.os.build.MODEL;
                //Get CPU ABI
       String cpu_abi = android.os.build.CPU_ABI;
                //Get Manufacturer
String manufacturer = android.os.build.MANUFACTURER;
                //Get Board
         String board = android.os.build.BOARD;
```

Read the /proc/cpuinfo file to determine the CPU Information.

```
    if (new File("/proc/cpuinfo").exists()) {
            try {
     BufferedReader br = new BufferedReader(new FileReader(new
    File("/proc/cpuinfo")));
   String aLine;
    while ((aLine = br.readLine()) != null) {
      sb.append(aLine + "\n");
     }
    if (br != null) {
       br.close();
    }
  } catch (IOException e) {
     e.printStackTrace();
  }
}
```

Similarly accelerometer, magnometer and gyroscope information can be determined by using the ***Sensory Manager*** class.

```
        SensorManager mSensorManager = (SensorManager)
    getSystemService(SENSOR_SERVICE);
                //Acclerometer
    msensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
                //Magnometer
    msensorManager.getDefaultSensor(Sensor.TYPE_MAGNOMETER);
                //Gyroscope
    mSensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE)
```

Total Available Memory can be determined using ***MemoryInfo*** and ***ActivityManager.***

```
        MemoryInfo mi = new MemoryInfo();
ActivityManager activityManager = (ActivityManager)
    getSystemService(ACTIVITY_SERVICE);
    activityManager.getMemoryInfo(mi);
long availableMegs = mi.availMem / 1048576L;
```

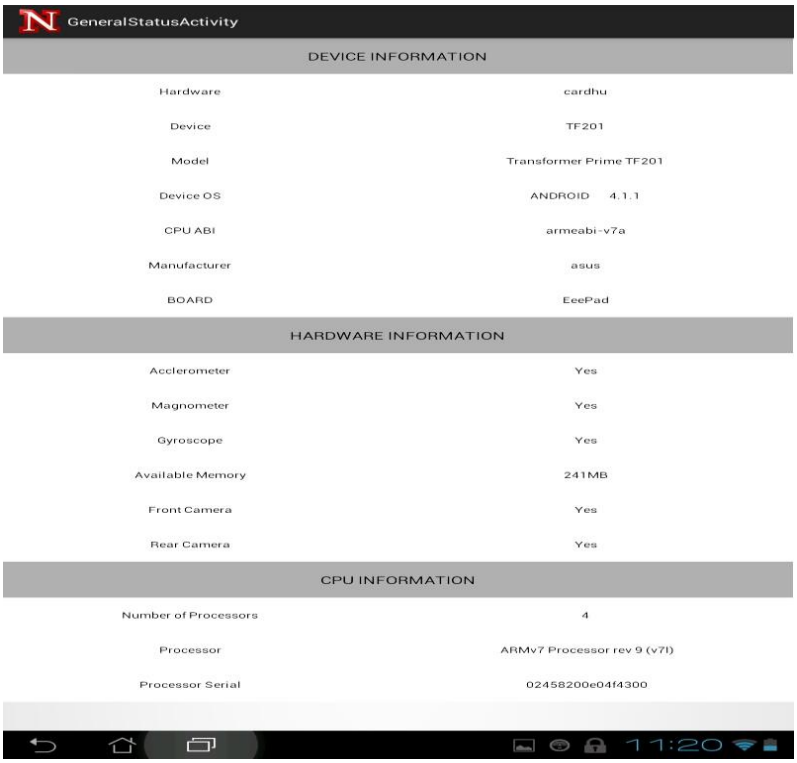

Figure 5: Hardware Information

4.3 Network

The ***Connectivity Manager*** monitors network connections (Wi-Fi, GPRS, UMTS, etc.), sends broadcast intents when network connectivity changes, attempts to "fail over" to another network when connectivity to a network is lost and provides an API that allows applications to query the coarse-grained or fine-grained state of the available networks.

```
ConnectivityManager connManager = (ConnectivityManager)
        getSystemService(CONNECTIVITY_SERVICE);

NetworkInfo mWifi =
connManager.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
```

If there is an instance of any Wi-Fi connection, then ***Wifi Manager*** is used to determine the list of configured networks, currently active Wi-Fi network, defines the names of various Intent actions that are broadcast upon any sort of change in Wi-Fi state and results of access point scans.

```
WifiManager wifimanager = (WifiManager)
getSystemService(Context.WIFI_SERVICE);
WifiInfo winfo = wifimanager.getConnectionInfo();
```

Network related information like MAC Address, IP Address, SSID, BSSID are determined as shown below

```
//Get mac address
mac_address = winfo.getMacAddress();

//Get IP Address
String ip_address = Formatter.formatIpAddress(winfo.getIpAddress());

//Get SSID
String ss_id = winfo.getSSID();

//GET BSSID
String bss_id = winfo.getBSSID();
```

Information such as Default gateway, DNS servers are determined by making use of the dhcpinfo class.

```
DhcpInfo d = wifimanager.getDhcpInfo();
//Subnetmask
String netmask =
Formatter.formatIpAddress(d.netmask);
//Default Gateway
String dg = Formatter.formatIpAddress(d.gateway);
//DNS Server 1
String dns1 = Formatter.formatIpAddress(d.dns1);
//DNS Server 2
String dns2 = Formatter.formatIpAddress(d.dns2);
```

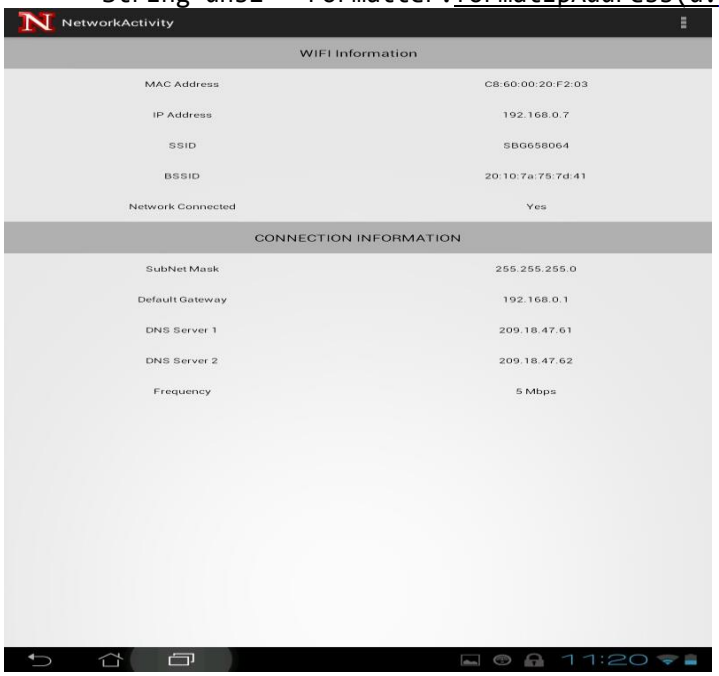

Figure 6: Network Information

4.4 Add New Location

The Add new location module allows adding information like location name, GPS coordinates and signal strength. The Google Location Services API, part of Google Play Services, provides a more powerful, high-level framework that automatically handles location providers, user movement, and location accuracy. Getting user location in Android works by means of callback indicating the intent to receive location updates from the ***LocationManager*** ("Location Manager") by calling requestLocationUpdates() and passing it a LocationListener. The LocationListener must implement callback methods that the Location Manager calls when the user location changes or when the status of the service changes.

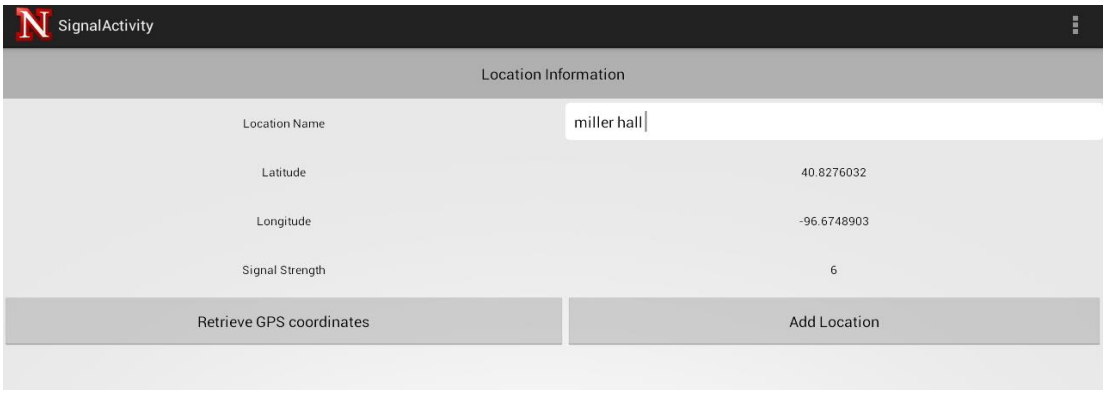

Figure 7: Add New location screen

```
// Get the location manager
        locationManager = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
        // Define the criteria how to select the location provider ->
use
        // default
        Criteria criteria = new Criteria();
        provider = locationManager.getBestProvider(criteria, false);
        location = locationManager.getLastKnownLocation(provider);
```

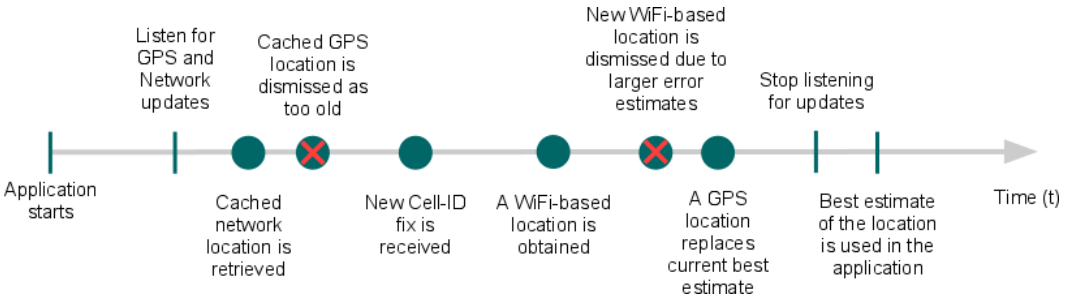

Figure 8: A timeline representing the window in which an application listens for location updates.

```
    /* Request updates at startup */
      @Override
      protected void onResume() {
        super.onResume();
```

```
          locationManager.requestLocationUpdates(provider, 400, 1,
this);
        }

      /* Remove the locationlistener updates when Activity is paused */
        @Override
        protected void onPause() {
          super.onPause();
          locationManager.removeUpdates(this);
        }

        @Override
        public void onLocationChanged(Location location) {
          lat = location.getLatitude();
          lng = location.getLongitude();
        }
```

The Wi-Fi signal strength is computed using the Wi-Fi Manager Class as shown below.

```
        //WifiManager Class
              WifiManager wifimanager =
      (WifiManager)getSystemService(WIFI_SERVICE);
        linkspeed = wifimanager.getConnectionInfo().getLinkSpeed();
```

After obtaining the location information and signal strength, we make use of SQLite DB to store the corresponding location in order to be plotted in the google map. A table named 'location' is created with the following attributes:

- Location_id : An unique key identifier for the location table.
- Location_name : Specifies the location name.
- Location_latitude : Specifies the location latitude.
- Location_longitude : Specifies the location longitude.
- Location_signal_strength : Specifies the Wi-Fi signal strength for the corresponding location.

4.5 Map

Generate a SHA-1 fingerprint using java key tool with the help of the following command at the terminal as show in Figure 9.

keytool -list -v -alias androiddebugkey -keystore "C:\XXX\XXX\.android\debug.keystore" -storepass android -keypass android

```
Alias name: androiddebugkey
Creation date: Apr 21, 2013
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Android Debug, O=Android, C=US
Issuer: CN=Android Debug, O=Android, C=US
Serial number: 8f4094e
Valid from: Sun Apr 21 20:38:01 CDT 2013 until: Tue Apr 14 20:38:01 CDT 2043
Certificate fingerprints:
         MD5:  4F:A7:7D:5C:4D:6E:67:B3:B4:54:B5:3E:09:EB:7A:A7
         SHA1: 9B:F1:FF:E3:8B:B8:B1:4B:FA:70:84:D0:38:64:94:63:7A:7A:05:7D
         SHA256: 67:8F:25:D1:AF:DC:82:31:A0:0F:B3:C4:7B:F6:5B:BC:49:3B:8D:9D:49:
3F:8C:5D:93:43:E3:36:12:A9:DA:9A
         Signature algorithm name: SHA256withRSA
         Version: 3
```

Figure 9: SHA-1 Fingerprint

To embed google maps in native android application, Google Play Services SDK needs to be installed as shown in Figure 10 and adding Google Play Services library to the source project.

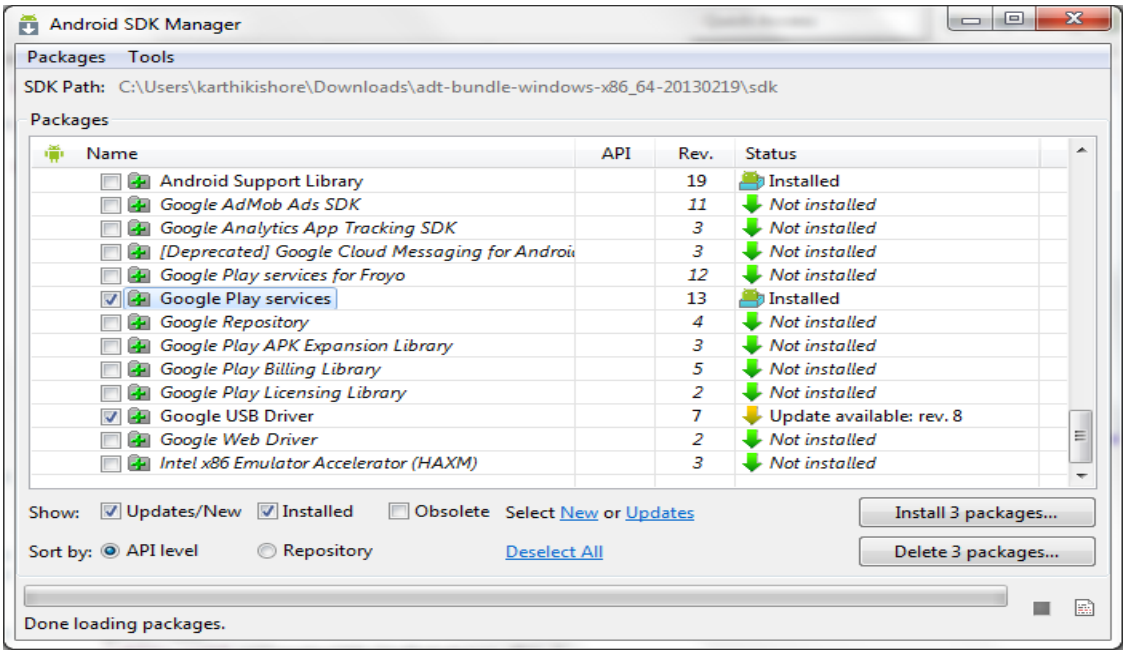

Figure 10 : Google Play Services Install

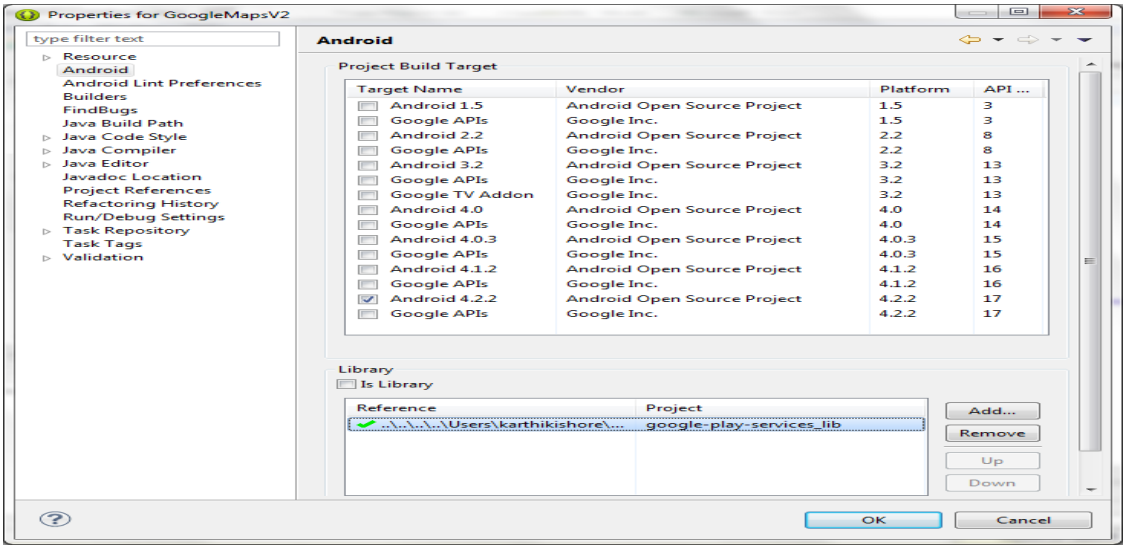

Figure 11 : Adding Google Play Services library

Once the application is imported successfully the google play services library, an API key for the google MAPS Android API V2 is created after the Google Maps Android API V2 in Services tab is enabled and create a new android key. Once the key is generated, it needs to be incorporated into the project (androidmanifest.xml) and add the sufficient permissions like ACCESS_NETWORK_STATE, INTERNET, WRITE_EXTERNAL_STORAGE, ACCESS_COARSE_LOCATION, ACCESS_READ_SERVICES, ACCESS_FINE_LOCATION and OPENGLESV2 as shown in Figure 16.

```
<meta-data
      android:name="com.google.android.maps.v2.API_KEY"
      android:value=" AIzaSyCFAZ_z_2a6GdRFEUbL8cpeICGFgLFwg8w" />
```

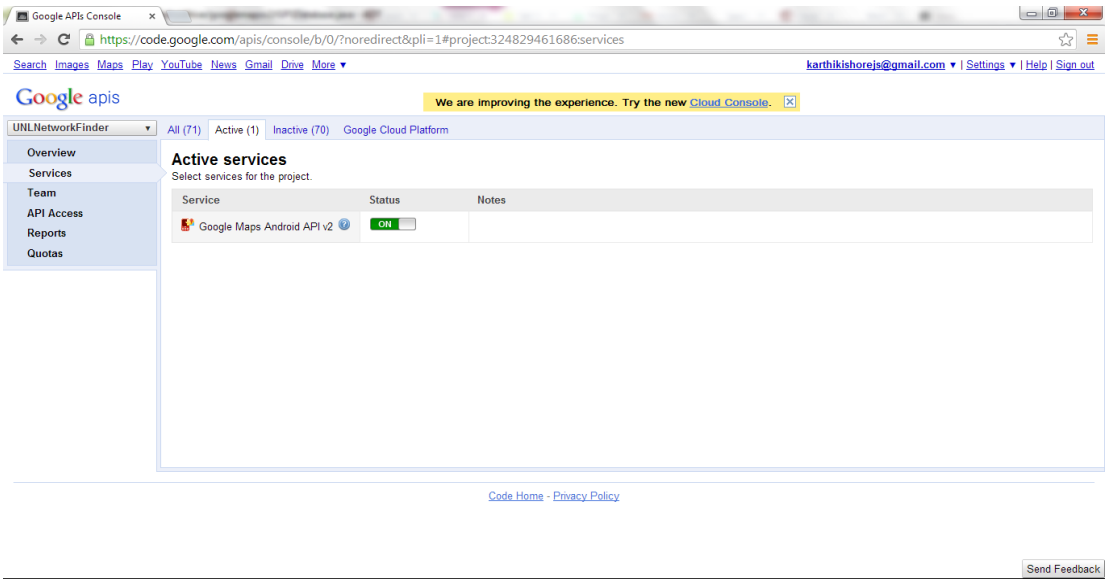

Figure 12: Enabling Google Maps API V2

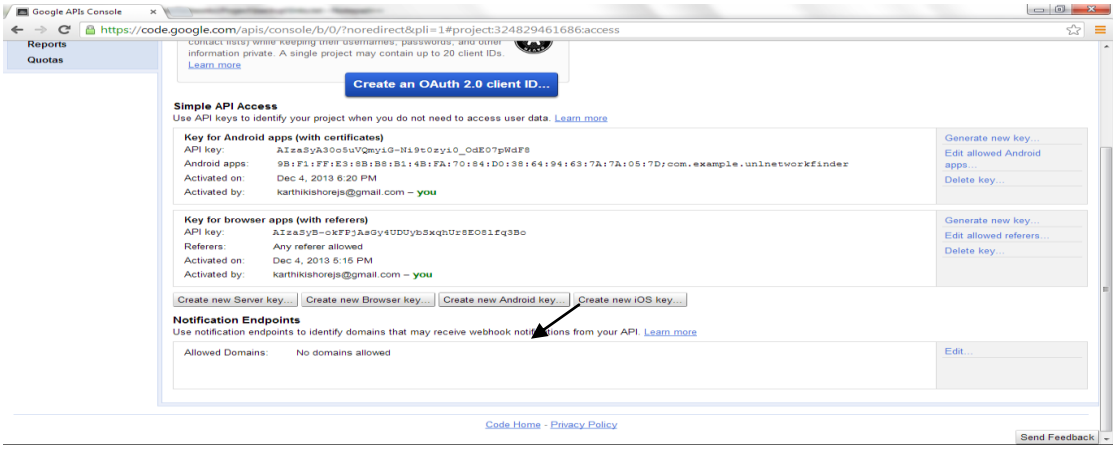

Figure 13 : Creating a new android key

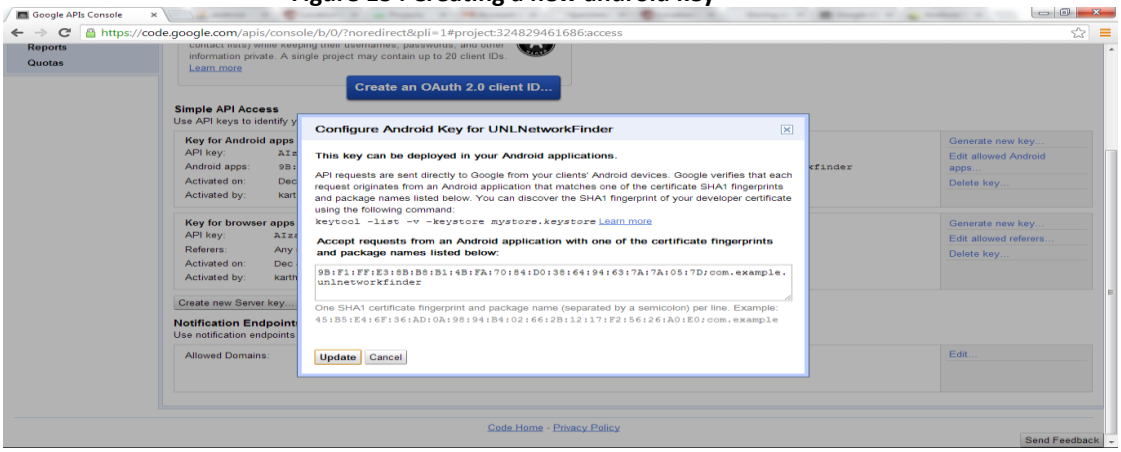

Figure 14: Append the source package name

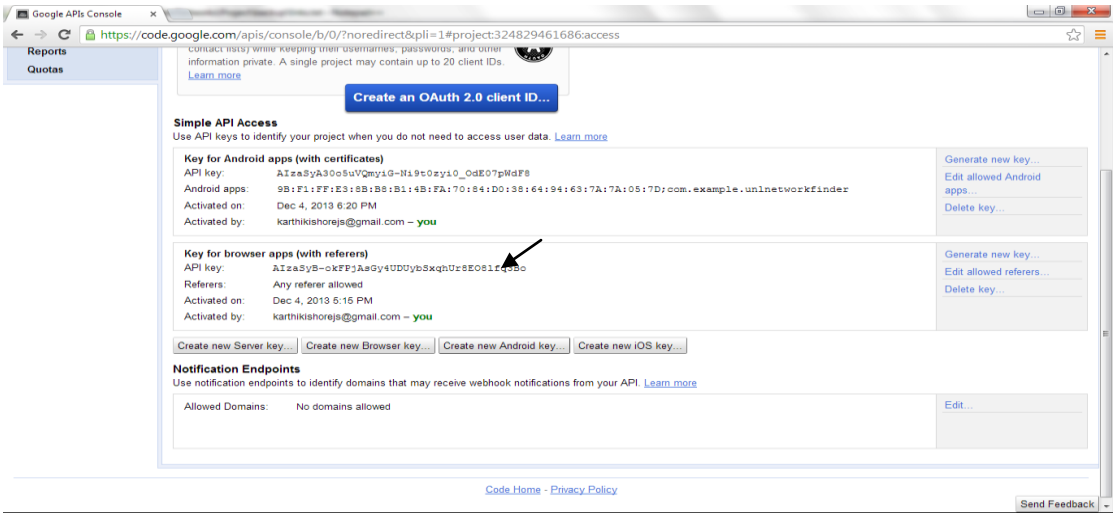

Figure 15: Key to be included in androidmanifest.xml

```
<permission
    android:name="info.androidhive.googlemapsv2.permission.MAPS_RECEIVE"
    android:protectionLevel="signature" />

<uses-permission android:name="info.androidhive.googlemapsv2.permission.MAPS_RECEIVE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>

<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

<!-- Required to show current location -->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

<!-- Required OpenGL ES 2.0. for Maps V2 -->
<uses-feature
    android:glEsVersion="0x00020000"
    android:required="true" />
```

Figure 16: AndroidManifest.xml permissions

Markers identify the locations on the map. The color of the markers depends on the computed Wi-Fi signal strength determined at various locations campus wide. Markers are objects of type Marker, and are added to the map with the ***GoogleMap.addmarker (markeroptions)*** method.

4.6 Speed Test Calculator

The speed of a wireless network depends on several factors particularly the protocols being used. The signal range a Wi-Fi wireless network supports also affects its overall speed. Like most kinds of computer networks, Wi-Fi supports varying levels of performance depending on which technology standards it supports.

A speed test measures the performance of a network during a (usually short) period of time. The tests normally send and receive data over the network and calculate performance according to the amount of data transferred and how much time was required. The measurement normally used for network speed is counted as the number of bits that travel across the internet. Here we determine the speed of the Wi-Fi network connection by checking the download speed of file placed in a remote server. The concept of threading is used where a slave worker actually does the computation and returns the result to the master. The size of the file placed in the remote server is in the order of 10.5 MB. The below figure shows the network speed calculated at some location across campus and calculated time was around~820 kbps.

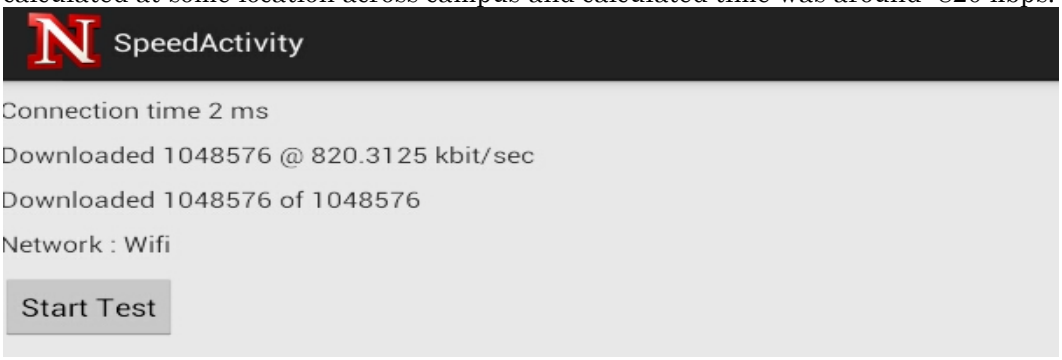

Figure 17: Speed Calculation across campus

```
String downloadFileUrl="http://cse.unl.edu/~kjakklis/index.php_bkp";
long startCon=System.currentTimeMillis();
URL url=new URL(downloadFileUrl);
URLConnection con=url.openConnection();
con.setUseCaches(false);
long connectionLatency=System.currentTimeMillis()- startCon;
stream=con.getInputStream();

Message msgUpdateConnection=Message.obtain(mHandler, MSG_UPDATE_CONNECTION_TIME);
msgUpdateConnection.arg1=(int) connectionLatency;
mHandler.sendMessage(msgUpdateConnection);

long start=System.currentTimeMillis();
int currentByte=0;
long updateStart=System.currentTimeMillis();
long updateDelta=0;
int  bytesInThreshold=0;

while((currentByte=stream.read())!=-1){
    bytesIn++;
    bytesInThreshold++;
    if(updateDelta>=UPDATE_THRESHOLD){
        int progress=(int)((bytesIn/(double)EXPECTED_SIZE_IN_BYTES)*100);
        Message msg=Message.obtain(
        mHandler, MSG_UPDATE_STATUS, calculate(updateDelta, bytesInThreshold));
        msg.arg1=progress;
        msg.arg2=bytesIn;
        mHandler.sendMessage(msg);
        //Reset
        updateStart=System.currentTimeMillis();
        bytesInThreshold=0;
    }
    updateDelta = System.currentTimeMillis()- updateStart;
```

Figure 18: Source code to determine Wi-Fi speed

5. RESULTS

We begin our analysis by classifying the Wi-Fi signal strength - those locations across campus that exhibited high signal coverage or low signal coverage and also a detailed picture on the Wi-Fi speed campus wide.

5.1 Coverage Map

The signal strength measured from a scale of 0-100. All those values less than 70 fall under the low signal coverage category and values greater than 70 fall under the strong signal strength coverage category. Therefore, by determining the values of signal strength at various locations, we get the overall signal strength picture across UNL campus. Figure 19 gives an outline view of the Wi-Fi signal strength information identifying the locations that has strong/weak Wi-Fi network signal.

5.1.1. UNL City Campus

In this section, we discuss on the signal strength for different locations in City Campus. Figure 20 shows the Wi-Fi signal information across UNL City Campus. The red and green markers indicate the instance of poor and strong signal strength respectively. The Wi-Fi signal strength varied significantly across different locations. For example, in one corner of the student union near Runza, good presence of Wi-Fi

signal strength was found, whereas when we move towards the Wells-Fargo Bank signal strength tends to drop and on reaching the campus bus stop, the signal strength varies drastically sometimes to the extent of zero and most cases in the range below 50. The reason attributed might be due to the fact that Wi-Fi connections can drop occasionally on devices located near the edge of the network's wireless signal range and Wi-Fi links generally become more unstable with distance.

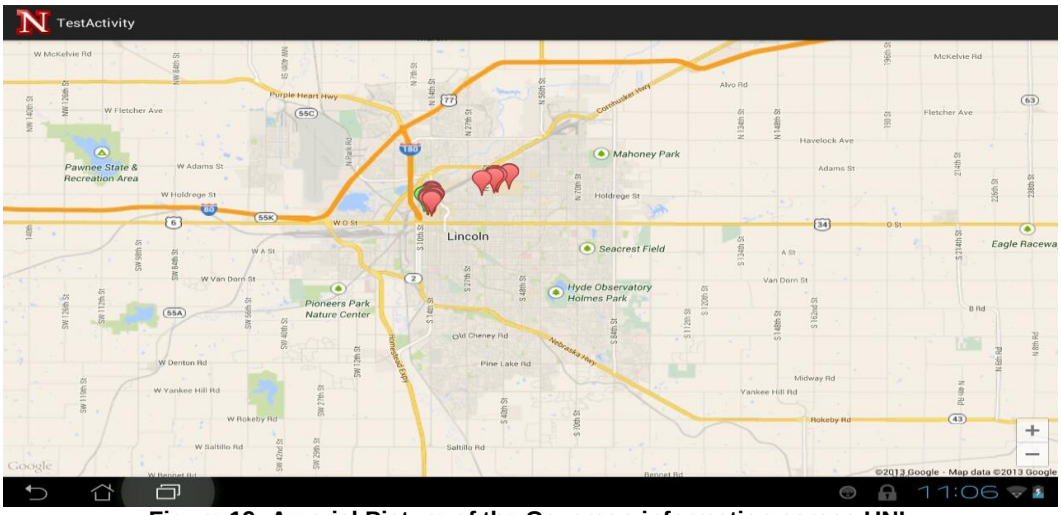

Figure 19: An arial Picture of the Coverage information across UNL

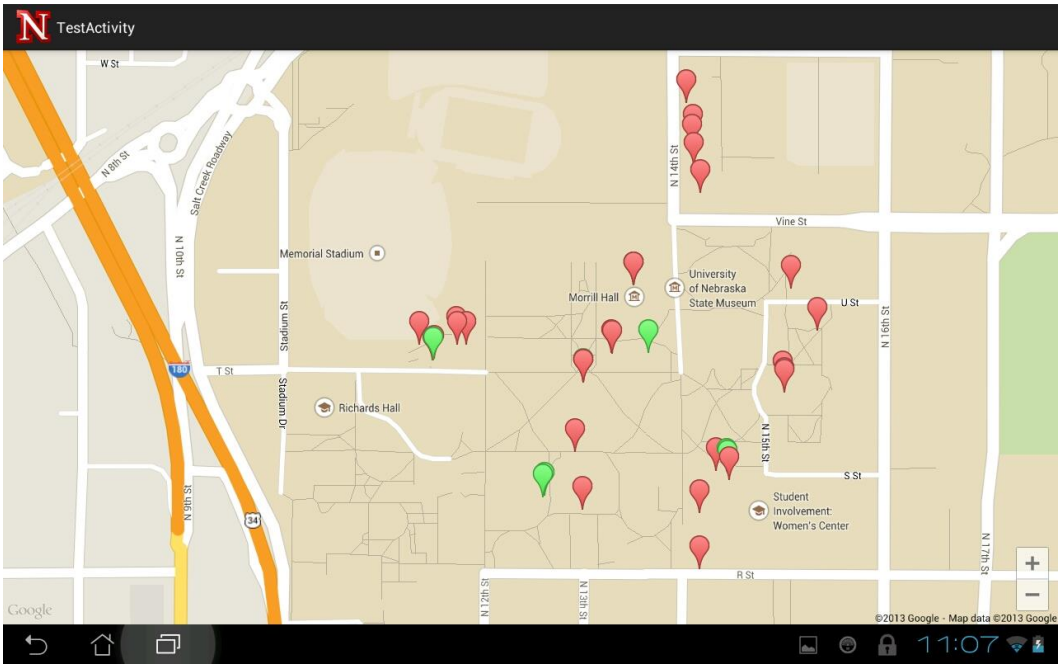

Figure 20 : Wi-Fi Coverage Information across UNL City Campus

Figure 21 : Wi-Fi Coverage Information at Avery Hall, UNL City Campus

Figure 21 shows the Wi-Fi signal strength near various classrooms at Avery hall. There seems to be non-uniformity because room 119 had high signal strength whereas math library in the basement had average signal strength.

5.1.2 UNL East Campus

In this section, we discuss on the signal strength for different locations in East Campus. Figure 22 shows the Wi-Fi coverage information across UNL East Campus. The red markers indicate the instance of poor signal strength. We observe that, there tends to be less/zero Wi-Fi signal in the east campus loop. Also places like car parking lots, water plants which had no traces of Wi-Fi signal. The main reason may be due to insufficient Wi-Fi Network Range & power and Wi-Fi Radio Interference.

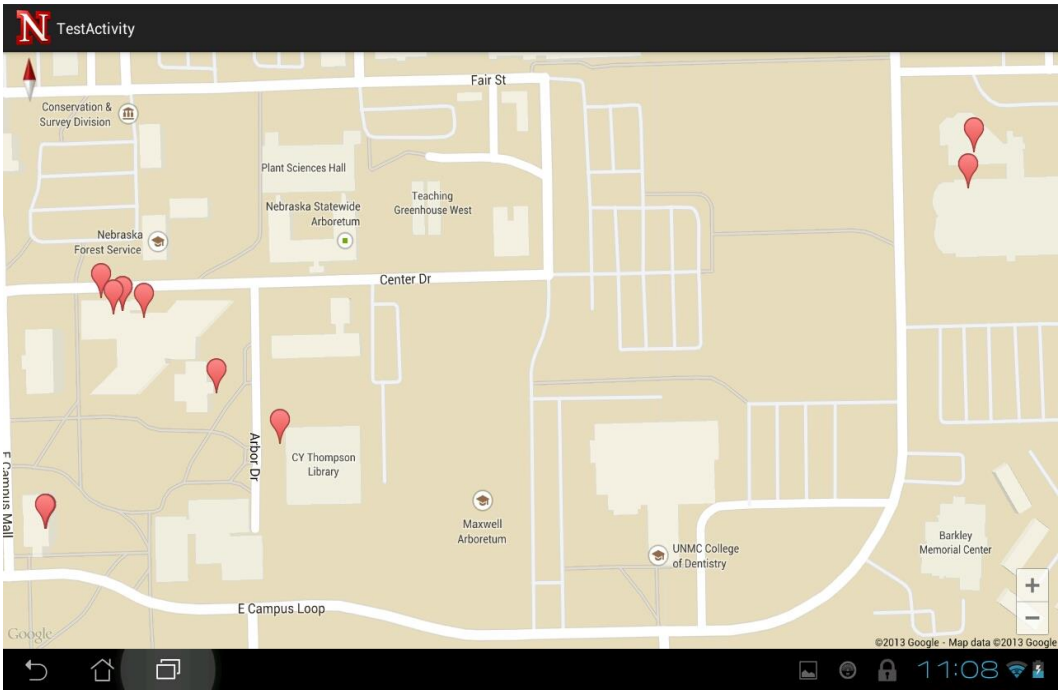

Figure 22 : Wi-Fi Coverage Information across UNL East Campus

5.2 Speed Test

Table1 shows the different Wi-Fi speed recorded at various locations across UNL City and East campus. The result shown in the table is an approximation of values taken at particular location. From the table, it can be inferred that Sellek hall had lower speed, while Henzlik hall had higher speed in UNL City campus. East campus student union, College of Dentistry, Fedde Hall and Leverton Hall had high speeds in the order of 800 kbps on UNL East campus. The low speed might be due to the overloading of the Wi-Fi access point or not mounting the router/access point close enough for best performance.

S.No		Speed (kbps)	S.No	Location Name	Speed in kbps
1.	Avery Hall 119	789.06	1.	Filley Hall	762.82
2.	Avery Hall 114	781.25	2.	Miller Hall	789.44
3.	Burnet Hall	781.25	3.	AgComm Building	771.51
4.	Andrews Hall	765.25	4.	Love Memorial Hall	800.01
5.	Morill Hall	735.2	5.	C.V. Thompson Library	742.27
6.	CBA Building	750.15	6.	Burr Hall	784.39
7.	Schorr Center	768.17	7.	Leverton Hall	801.21
8.	Love Library South	795.49	8.	Entomology Hall	775.28
9.	Love Library North	784.45	9.	Keim Hall	721.82
10.	Mablee Hall	751.24	10.	Agricultural Hall	765.13
11.	Henzlik Hall	832.23	11.	Biochemistry Hall	721.48
12.	Sellek Quad Building D	723.12	12.	Forestry Hall	789.12
13.	Jorgenson Hall	781.43	13.	Chase Hall	734.56
14.	Seaton Hall	800.75	14.	College of Dentistry	831.93
15.	Memorial Stadium	805.2	15.	Fedde Hall	803.67
16.	Student Union	725.39	16.	East Union	811.52

Table 1 : Wi-Fi Speed reported at different locations across UNL City and East Campus

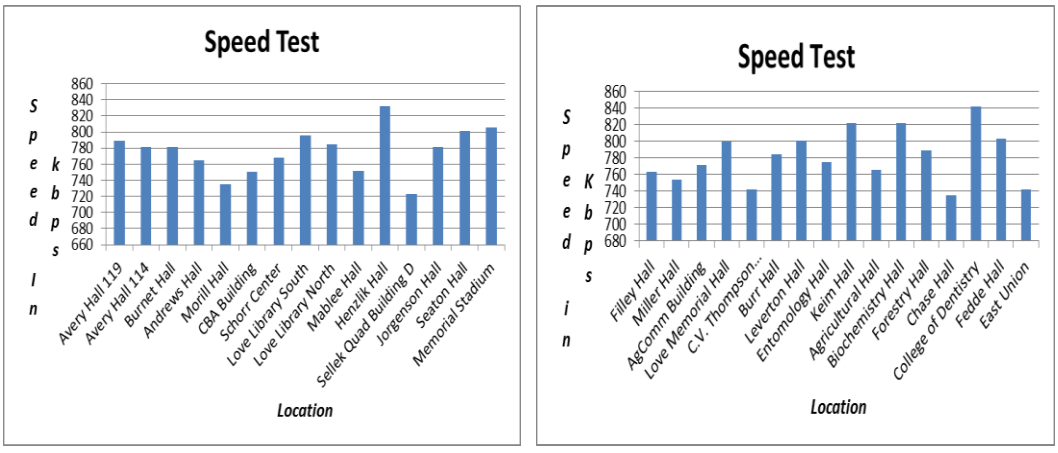

Figure 23: Wi-Fi speeds at various locations across UNL City and East Campus

6. CONCLUSION & FUTURE WORK

The Android application pretty much was able to identify locations that had good or average strength campus-wide. It served to find certain interesting aspects like the absence of Wi-Fi signal across the East campus loop, car parking lots etc. The Speed test served to identify locations that high, average and low speed connections. It showed that the variable Wi-Fi signal strength may be due to Insufficient Wi-Fi Network range & power, Wi-Fi Radio interference and weak or non-functional Wi-Fi access points.

The Future scope lies in extending the application with respect to the scalabilty of the location and mobile 4G network. A more clinical hybrid model combining the Wi-Fi and 4G network connection can yield some powerful and significant results for both the user and the mobile carrier.

The Source code of the application can be accessed from Git Hub.
https://github.com/karthikishorejs/UNLNetworkFinder

REFERENCES

1. http://developer.google.com
2. http://compnetworking.about.com/od/wirelessfaqs/f/signal_strength.htm
3. http://www.tutorialspoint.com/android/android_architecture.htm
4. http://home.deib.polimi.it/bellasi/lib/exe/fetch.php?media=students:tienthanh_finalreport.pdf
5. http://compnetworking.about.com/od/internetaccessbestuses/a/how-to-check-network-connection-speed.htm
6. http://www.androidhive.info/2013/08/android-working-with-google-maps-v2/
7. http://www.androidhive.info/2011/11/android-sqlite-database-tutorial/
8. http://www.javabeat.net/location-tracker-in-android-using-gps-positioning-and-sqlite/