Wireless Fingerprinting

Jared F. Bennatt

University of Texas at San Antonio

fts180@my.utsa.edu

December 9, 2014

Overview

Goals

- Find geographic location of WiFi access points at UTSA.
- Find MAC Address (BSSID)[1] of each access point.
- Find channel of each access point.

[1] BSSID: **B**asic **S**ervice **S**et **I**dentifier (which is usually the MAC address of the router).

Background

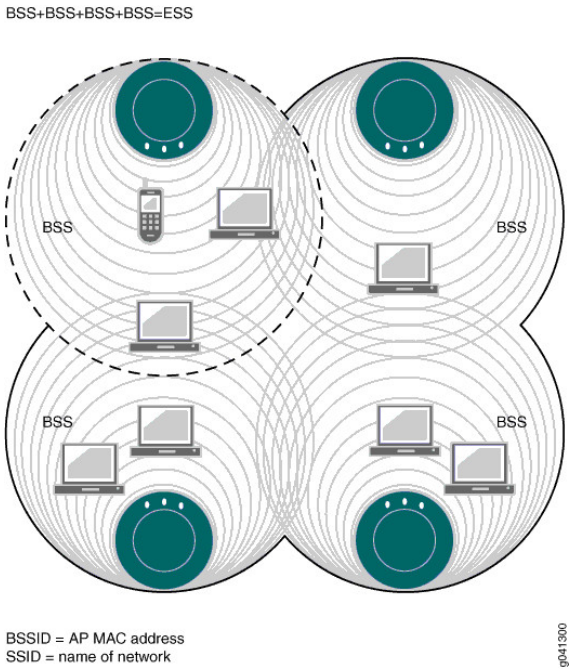
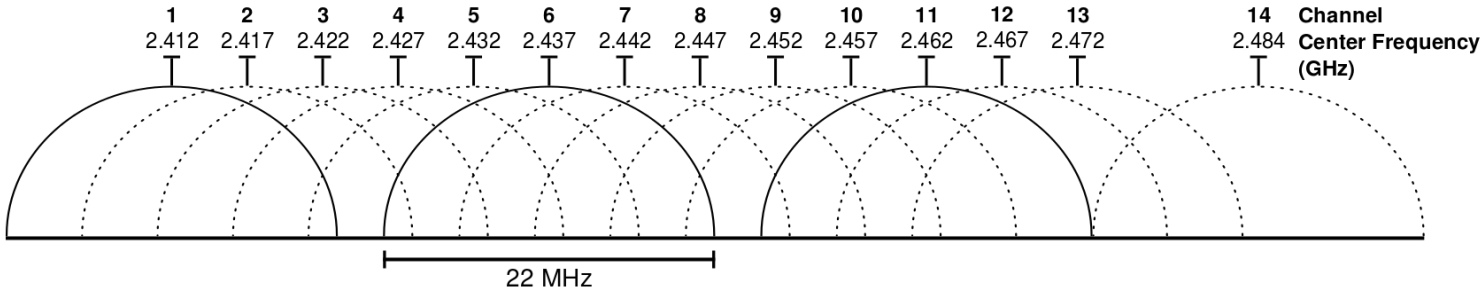


Figure : from Understanding the Network Terms SSID, BSSID, and ESSID

- WLAN (**W**ireless **L**ocal **A**rea **N**etwork) is made up of several BSS's (**B**asic **S**ervice **S**et).
- As you move, while you stay connected to the "same" network, you may connect to a *different* router (BSS).

Channels



- Channels are defined by their carrier frequency (e.g. channel 6 is carried on a frequency of $2,437$ MHz).
- US allows only channel 1-11 for 802.11b/g/n.

WIFI Tracker

- Use Android phone (Samsung Galaxy S2 with Android Jelly Bean 4.1.2).
- Detect network changes.
- Fire GPS locator when network change is detected.
- Collect GPS location along with previous and current network.
- Detect boundaries of various access points and create map based on boundaries.

Problems WIFI Tracker

- Biggest Problem: GPS isn't readily available and almost impossible to get inside a building (I was never able to find my GPS location inside the NPB building).
- Network state doesn't change unless network becomes unavailable (e.g. at the bus stop by the music building or near the roadrunner statue near JPL).
- Need to check *more* than just network state, but *also* supplicant state.
- Poor programming: I was getting way too many network change notifications and it was difficult to handle this fact while GPS location was coming in much slower.

WiFi Meter

- Implemented in Android (compatible with Froyo, using Samsung Galaxy S2 with Android Jelly Bean 4.1.2)
- Forget about GPS, just find location of access points by building and room number (or thereabouts).
- Use RSSI (**R**eceived **S**ignal **S**trength **I**ndication) to hone in on access point.
- Android `ScanResult.level` gives RSSI in dBm (decibel-milliwatts).
- Not necessary to be connected to network to find access point.

dBm

- dBm: decibel-milliwatts.
- power ratio of dB to a reference milliwatt .
- In radio and telephony, usually referenced to 600 Ω.
- In radio frequency (e.g. wireless networks), usually referenced to 50 Ω.
- Log scale, an increase of 3 dBm is roughly equivialant to doubling the power.[2]

$$\text{dBm} = 10 \log_{10} \left(\frac{P}{1 \text{ mW}} \right) \qquad P \equiv \text{power output in mW}$$

[2] $10 \cdot \log_{10}(2x) = 10 \cdot (\log_{10}(2) + \log_{10}(x)) \approx 3.01 + 10 \cdot \log_{10}(x)$

Problems with WiFi Meter

- RSSI does *not* directly relate to distance from router.
- RSSI depends on distance *and* obstacles (e.g. I get a smaller signal in my kitchen even though it's closer to the router but separated by a wall).
- Possible Application? Might be able to detect the structure of a building using tomography techniques via the WiFi signal strength and knowledge of the location of the routers and their power output.

Implementation: WifiMeter

```
14 public class WiFiMeter extends BroadcastReceiver {
15
16     private static final IntentFilter FILTER = new IntentFilter(
17             WifiManager.SCAN_RESULTS_AVAILABLE_ACTION);
18
19     private final Activity main;
20     private final WifiManager wMan;
21     private List<ScanResult> scanResults;
22
23     private volatile boolean scanning = false;
24
25     public WiFiMeter(final Activity main, final WifiManager wMan) {
26         this.main = main;
27         this.wMan = wMan;
28     }
29
30     @Override
31     public void onReceive(Context context, Intent intent) {
32         scanResults = wMan.getScanResults();
33
34         scanning = false;
35     }
36
37     public List<ScanResult> scan() {
38         main.registerReceiver(this, FILTER);
39         scanning = true;
40         wMan.startScan();
41         while (scanning)
42             SystemClock.sleep(100);
43
44         main.unregisterReceiver(this);
45
46         return scanResults;
47     }
48
49 }
```

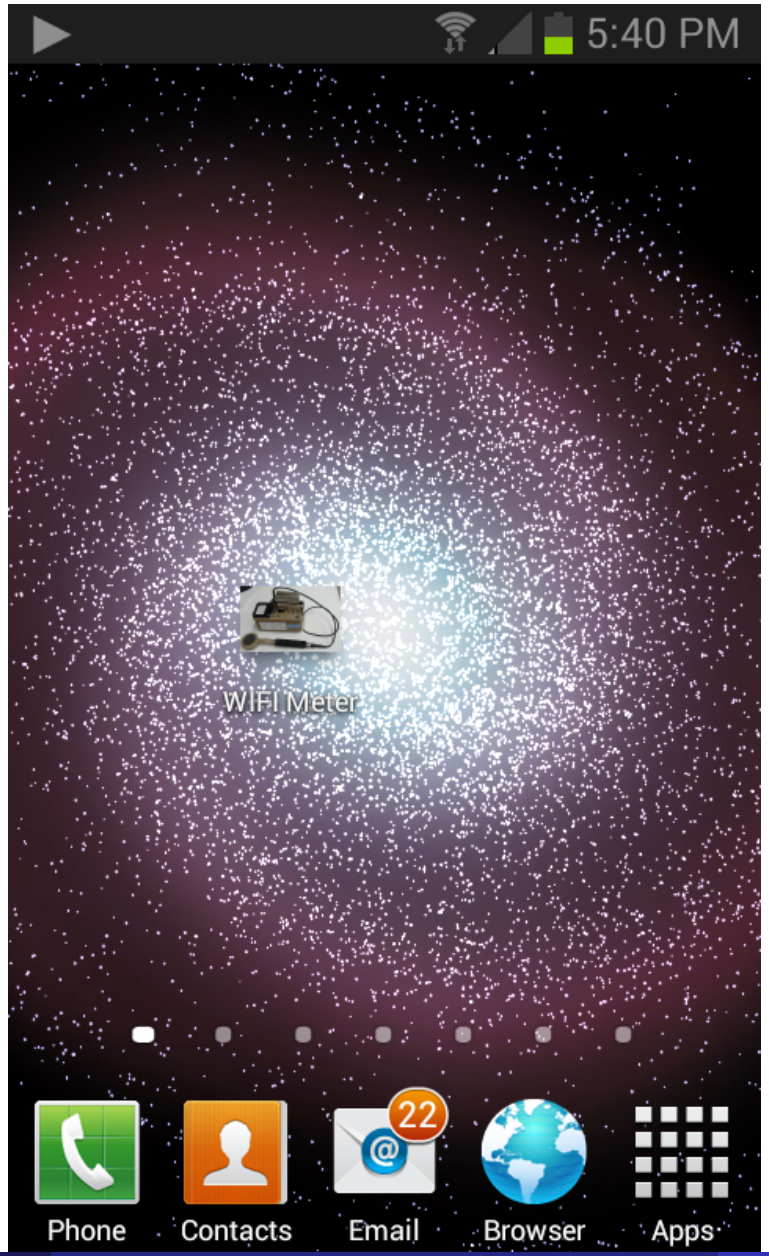
Implementation: WifiMeterUI

```
153    private ScanResult getScanResult() {
154        final WifiInfo wInfo = wMan.getConnectionInfo();
155        targetBSSID = wInfo == null || targetBSSID != null ? targetBSSID
156                : wInfo.getBSSID();
157
158        if (targetBSSID == null) {
159            return null;
160        }
161
162        // else return results
163        for (final ScanResult result : results)
164            if (targetBSSID.equals(result.BSSID))
165                return result;
166
167        return null;
168    }
```

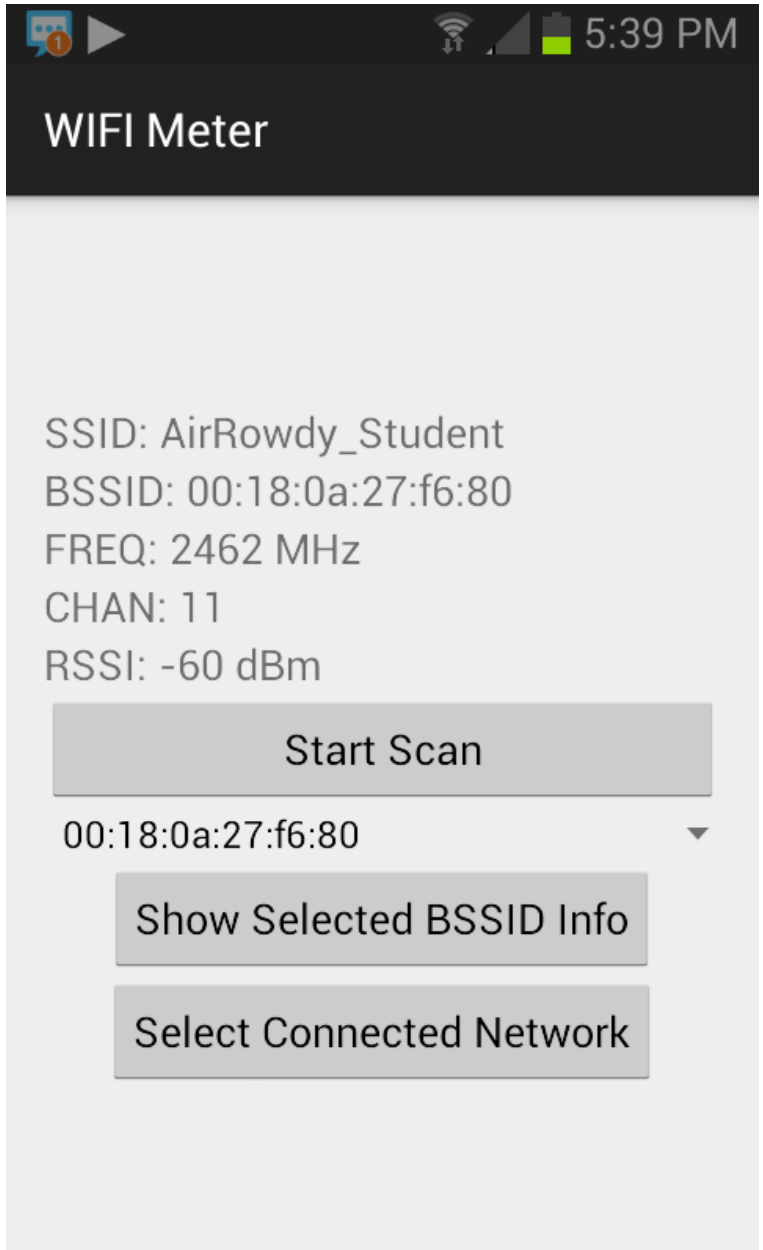
Implementation: WifiMeterUI (cont.)

```
117             final String SSID = "SSID: " + result.SSID;
118             final String BSSID = "BSSID: " + result.BSSID;
119             final String FREQ = "FREQ: " + result.frequency + " MHz";
120             final String CHAN = "CHAN: " + getChannel(result.frequency);
121             final String RSSI = "RSSI: " + result.level + " dBm";
122
123             main.runOnUiThread(new Runnable() {
124
125                 @Override
126                 public void run() {
127                     ssid.setText(SSID);
128                     bssid.setText(BSSID);
129                     freq.setText(FREQ);
130                     chan.setText(CHAN);
131                     rssi.setText(RSSI);
132                 }
133             });
```

WiFi Meter Icon



WiFi Meter UI



- Found 20 routers in NPB building (and I'm sure I missed several).
- Had visual confirmation for all but 6 routers.
- Most routers use channell 11 (2.462 GHz). Channel 1 and channel 6 were the only other two observed (except 5 GHz router described below).
- Virtually all BSSIDs for start with prefix 00:18:0a.
- Router in NPB 2.124 had BSSID 02:18:1a:26:83:06 and operated on channel 149 (5.745 GHz) meaning that it was using the IEEE 802.11a/h/j/n/ac standard.

Questions???