# A Wireless Communications Radio Frequency Scanner for Signal Measurement

**<u>ECE 4012 Senior Design Project</u>**

**Section L3B, Wireless RF Scanner Team (GD3)**

**Project Advisor, Dr. Greg Durgin**

**Sahil Gupta, sahilgupta05@gatech.edu**

**Cameron Karlsson, ckarlsson3@gatech.edu**

**Avnish Kumar, akumar96@gatech.edu**

**Pooja Modi, pmodi6@gatech.edu**

**Vatsal Patel, vpatel92@gatech.edu**

**Prahlad Venkatesh, pvenkatesh3@gatech.edu**

**Date Submitted**

**May 1st, 2014**

## Executive Summary

The team has designed a wireless communications scanner that will be used in a mobile environment for measuring the RF performance of mobile telephone networks. This device has been prototyped for a company called DasPoint Inc., which uses these transceivers to optimize distributed antenna systems. DasPoint has focused on designing distributed antenna systems for open areas such as football stadiums, college campuses, building complexes etc. It uses these scanners to scan the signals received by these antennas and thereby test its design of distributed antenna systems. The current device they use is bulky and is heavy to carry around to perform vehicle-based or walk-based tests. This product integrates various components needed for the scanner and transmits all the results on an Android device. This makes the whole testing process less bulky and more real time. It also provides a much more user-friendly interface to the people conducting these tests, and will give them much more control and debugging tools on-site while they are conducting the tests.

The scanner consists of three major components, namely, Android mobile application, control system, and, the software-defined radio receiver. The Android interface is the first point of communication through which the user enters the frequency and bandwidth of interest. These parameters are then communicated to the control system through a Wi-Fi connection. The control system requests the software-defined radio, which contains the antenna, to read the signal in the user-specified bandwidth and frequency. After this, the control system runs different algorithms on the data to generate final results. Currently, the algorithms include signal measurement algorithms for UMTS and Fast Fourier transform of the complex I-Q values generated by the software-defined radio. Finally the Android application will display the data in a simple user interface [14].

The device costs about $1500, which includes the cost of the parts and development. The market value for the device will be about $8000. The cost of the device is about the same as the current products in the market, but efficiency of the device will save companies a lot of cost. Using this device requires less manpower and time to analyze a specific network; therefore, reducing operating costs significantly.

**Table of Contents**

# A Wireless Communications Radio Frequency Scanner for Signal Measurement

## 1. Introduction

The Wireless RF Scanner team has designed a scanner that tests the strength of a mobile network by scanning signals in a given frequency and bandwidth, and displaying the relevant statistics. It serves as a portable and an affordable way for companies to measure and optimize wireless signals. The team has worked with DasPoint to gather the necessary parts for the project. The total cost of the parts was about $1500.

### 1.1 Objective

The team has designed and prototyped a device that can tune to frequencies from 300MHz to 3.8GHz and gather data to analyze a specific network. The control system requests the wireless signals from the software-defined radio and processes it using UMTS analysis algorithms. These algorithms output specific parameters such as ECIO, sample number, time, etc. that help the end user gather valuable data about the network status. This data is then displayed in the form of plots and lists on an Android platform based mobile application.

### 1.2 Motivation

The motivation for this project came from the fact that the team has designed a prototype product that will be used in the industry for the purpose of testing the strength of mobile networks and setting up distributed antenna systems. Working with a company that already has a strong presence in the industry assured the team that the scanner will be used in the real world. The device offers companies an easy way to analyze networks since it will require little training because of the simple user interface. This device saves companies time and money, as opposed to using more complex tools that require a trained users to operate.

## 1.3 Background

DasPoint, Inc. has developed a scanner that they currently use for testing. There is not much competition in the market due to small demand for such services. DasPoint has focused on designing distributed antenna systems for open areas such as football stadiums, college campuses, building complexes etc. It uses these scanners to test its design of distributed antenna systems. The device is based off the current device they use, which is bulky, slow, and hard to operate. The team has aimed to fix all those problems by using newer and faster technology.

The scanner consists of three major components, namely, Android mobile application, control system, and, the software-defined radio receiver. The Android interface is the first point of communication through which the user enters the frequency and bandwidth of interest. These parameters are then communicated to the control system through a Wi-Fi connection. The control system then requests the software-defined radio, which contains the antenna, to read the signal in the user-specified bandwidth and frequency. After this, the control system runs different algorithms on the data to generate final results. Currently, the algorithms include signal measurement algorithms for UMTS and Fast Fourier transform of the complex I-Q values generated by the software-defined radio. Finally, the results will be sent via Wi-Fi connection to an Android application, which will display the results in a simple user interface.

## 2. Project Description and Goals

The team has designed and prototyped a working device that gives signal statistics and spectrums based on a given frequency and bandwidth. The device comprises of an Android interface through which the user can enter the frequency and bandwidth of interest. A Linux operating system serves as a control system/mechanism that stores and sequentially triggers all the algorithms that handle the performance of the scanner.

- Dynamically tune anywhere from 300MHz to 3.8 GHz
- Dynamic bandwidth selection from 50KHz to 20 MHz
- RF Snapshot recorder capability
- Performs UMTS decoding – In the future, GSM/CDMA/LTE decoding
- Signal measurements such ECIO, sampling number, normal correlation, CPICH, RMS
- Android-based control with API for further development
- In the future - Using a GPS module provide accurate timing and location data with outputs

## 3. Technical Specifications

### 3.1 Quantitative Specifications

The table below contains the expected values versus the measured values for various specifications of our project.

**Table 1.** Technical Specifications

| Feature | Expected Value | Measured Value |
|---|---|---|
| Frequency Range | 300MHz - 3.8GHz | 300Mhz - 3.8GHz |
| Weight | 350g | 1200g |
| Battery Use Time | 6 hours | 10 hours |
| Distance Range | 0.5 miles | 2 miles |
| Size | 5” x 4” x 4” | 7.56” x 11.8” x 1.2” |
| Processing Speed | 1.6GHz | 2GHz |
| Data Range | 10m | 100m |

### 3.2 Qualitative Specifications

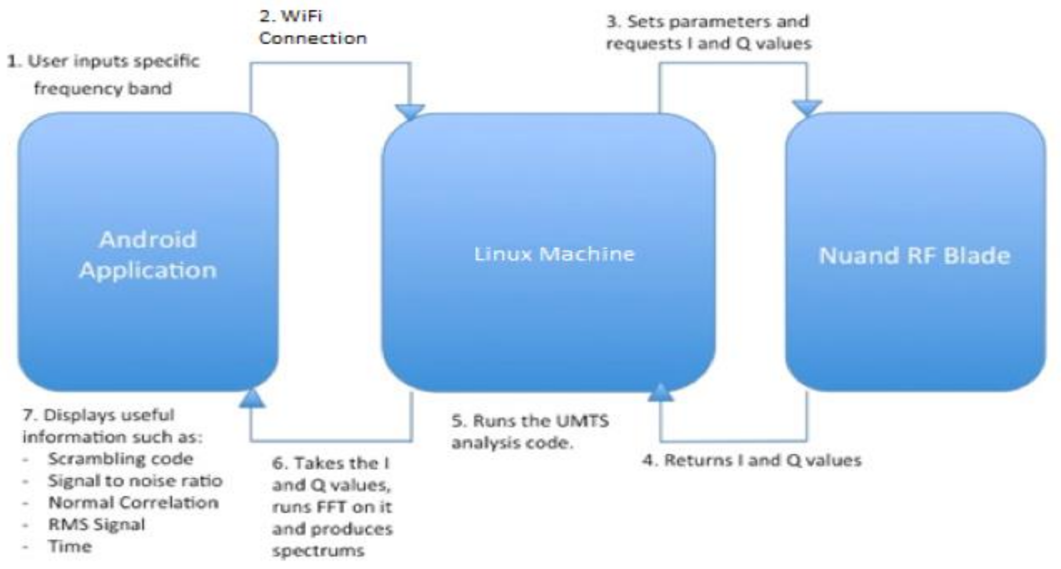
Initially intuitive menu screens, easy to find error faults on Android, easy to interpret results, and a protective case for the system were desired. All of these specifications were integrated into the project except for the protective case.

## 4. Design Approach and Details

### 4.1 Design Approach

*System Overview*

The scanner consists of three major components, namely, Android mobile application, control system, and, the software-defined radio receiver. The Android application serves as the user interface through which the user can enter the frequency and bandwidth of interest as well as access real time information on the specified signal such as the spectrum of the raw data, ECIO, and CPICH. The control system has all the algorithms that run the entire prototype. It first triggers the software-defined radio that has an antenna, to scan the signals in the desired frequency and bandwidth. Following that, it runs the signal processing algorithms and feeds the results into the Android application. The following figure shows the hardware block diagram of the entire system and how they interact with each other.

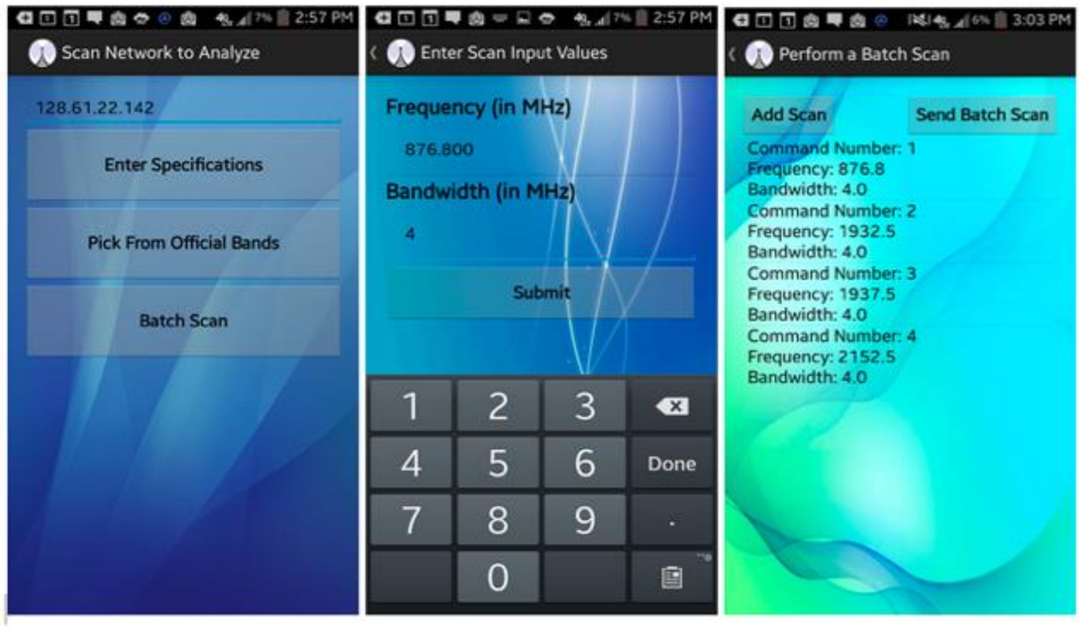


2. WiFi Connection
3. Sets parameters and requests I and Q values
1. User inputs specific frequency band
Android Application
Linux Machine
Nuand RF Blade
7. Displays useful information such as:
- Scrambling code
- Signal to noise ratio
- Normal Correlation
- RMS Signal
- Time
5. Runs the UMTS analysis code.
4. Returns I and Q values
6. Takes the I and Q values, runs FFT on it and produces spectrums

**Figure 1. Hardware block diagram of the entire system.**

*Android Application*

The Android application serves as the main interface through which the user interacts with the scanner and inputs the frequency and bandwidth of interest [6, 10]. It runs on the smartphone of the user and communicates with the control system through a Wi-Fi connection. This application has been developed on Android 4.4.2 KitKat using java. After receiving the frequency and bandwidth from the user, the Android application returns a spectrum of the FFT of the raw data using a line graph, along with a list of useful information such as CPICH, signal to noise ratio, and the RMS signal. The line graph has been developed using an external support library known as GraphView for Android. [6]



Command Number: 1
Frequency: 876.8
Bandwidth: 4.0
Command Number: 2
Frequency: 1932.5
Bandwidth: 4.0
Command Number: 3
Frequency: 1937.5
Bandwidth: 4.0
Command Number: 4
Frequency: 2152.5
Bandwidth: 4.0

**Figure 2. GUI of the Android application.**

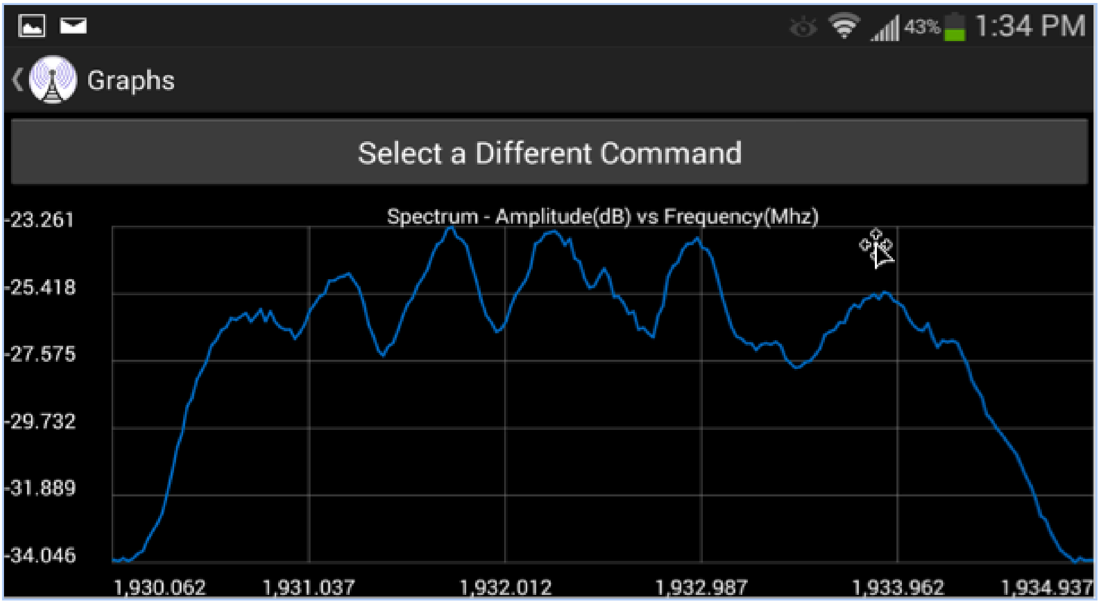


**Figure 3. An example plot of the FFT of the raw signal (amplitude (dB) vs frequency (MHz)).**

*Control System*

A Linux operating system serves as a control mechanism that stores and sequentially triggers all the algorithms that handle the performance of the scanner. The following figure shows the block diagram of the control system and the three main logic blocks through which it manages the device:

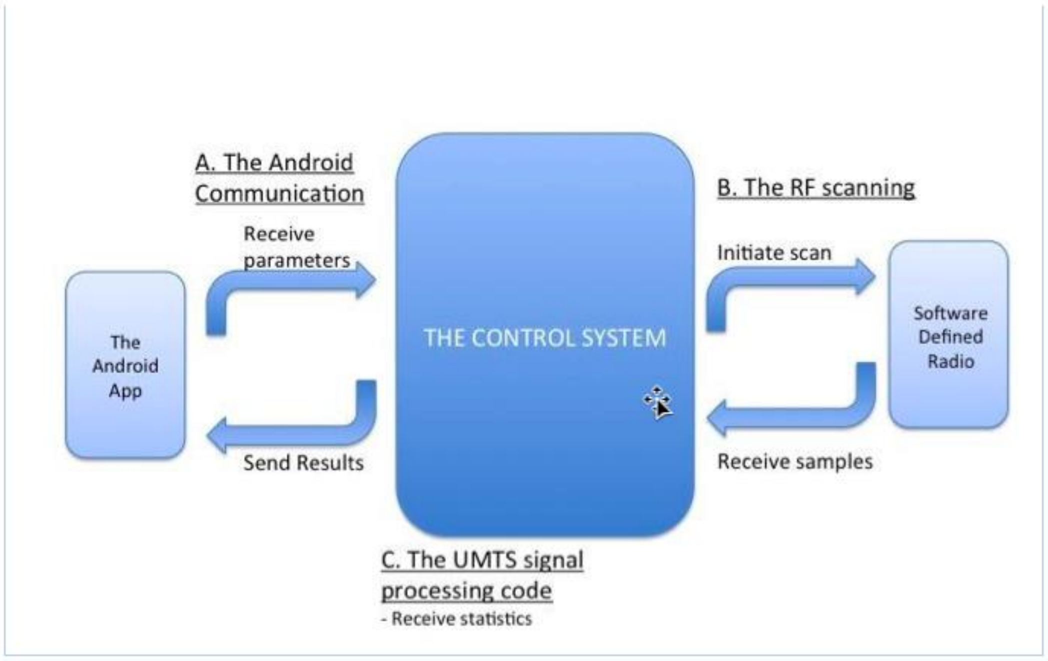


**Figure 4. Control system block diagram.**

A. The Android Communication – The control system communicates with the Android application through a wireless Wi-Fi connection. This communication is based on the client-server model, where the server is the Apache running on the control system and the Android is the client [12, 13]. The Android app makes restful API calls to the server by submitting a post request with the serialized string of scan parameters, namely frequency and bandwidth. Upon receipt of this string, the server de-serializes the string and launches a hidden console through which the RF Scanning code is triggered and the scan parameters are passed as inputs. The console then waits till the scanning and signal processing is complete and the results, namely signal statistics and spectrums, are returned in a RF Snapshot packet. These results are then serialized using JSON libraries and communicated to the Android Application for display to the user.

B. The RF Scanning Code – This C++ based algorithm is the core of the scanner's functionality and is triggered by the console when a post request is received from the scanner. For every request, the RF Scanning Service is activated. Steps below explain at the high level how this service works:

· First, it calls runSingleScan() function that communicates with the Nuand BladeRF through the USB port and initiates the scan in the given frequency and bandwidth. This function returns the samples containing I-Q values [2, 3, 4, 14]. Note that I corresponds to the in-phase component, while Q is the quadrature ($90^0$ phase shifted) part of the raw signal.

· Second, it calls processUMTSSamples() which starts the execution of the UMTS analysis code (explained next) that performs all the signal processing on the samples and returns signal statistics such as signal to noise ratio.

· Third, it calls getSpectrum() that performs a fast Fourier transform on the raw data and generates useful spectrums that, when displayed, makes it possible for the user to compare signal strength or spectrum noise between two locations [11].

· The statistics and spectrums serve as results which are packed into a snapshot response object and sent to the console after serializing them into the JSON format, which is a flexible text representation of objects [5, 15, 16] and helps in easy communication between object oriented programming languages, namely C++ and Java in our case. If multiple scans were requested, the application will append the results from each scan before sending.

C. The UMTS Signal Processing Code – The objective of the UMTS signal processing code, as mentioned earlier, is to provide useful information about the raw signal, like the CPICH and the signal to noise ratio [1]. The frequency and bandwidth specified in the processUMTSSamples() function are taken in by the UMTS signal processing code, and for each of the frequencies specified in the band, two primary functions – cell_search() and decode_layer_3() are performed. The purpose of cell_search() is to find the location of the time slots and return useful information like the location of the peak. In essence, this is where the bulk of the work is done. The raw signal is taken in and correlated against the primary secondary synchronizations (synchs). Then, since there are 512 scrambling codes and 64 secondary synch sequences, the number of possible scrambling codes is reduced to 8. Finally, all of these 8 scrambling codes are treated as ‘candidate’ scrambling codes, and the one with the highest peak is determined as the desired scrambling code (also known as CPICH). In decode_layer_3(), the raw signal is taken in as a parameter, upsampled, and then converted to a bitstream. The main purpose of this function is to return layer 3 information like cell ID, which is characteristic of each cell, and this is done by decoding the bitstream.

*Software-defined Radio*

In this project, the Nuand BladeRF is used as the software-defined radio, since it has very good processing power, has USB 3.0 support and is relatively cheap for its functionality. It has the antenna connected to it and its main function is to scan for the signals in the specified frequency and bandwidth. It outputs the I-Q values to the control system/Linux when it requests them.

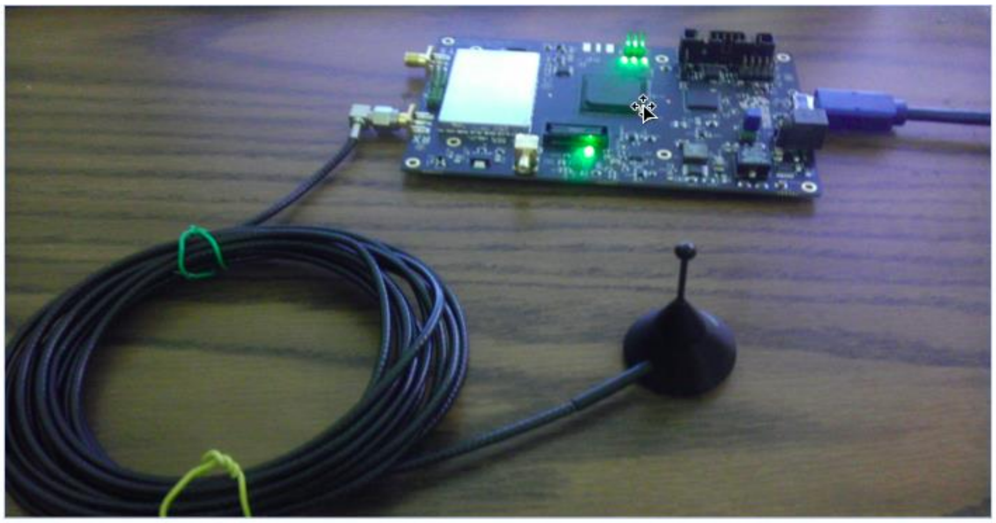


**Figure 5. Nuand BladeRF with antenna to scan for I-Q values of raw signal.**

## 4.2 Codes and Standards

There are several codes and standards that apply to the project:

1. Nuand bladeRF captures radio frequencies of various network protocols. This scanner only captures UMTS. However, other protocols like LTE, EDGE, HSPA, HSPA+, and CDMA can be implemented to the scanner to further the compatibility [1]. UMTS signal was captured to analyze and optimize the strength of the signal. UMTS Signal strength and noise ratio was found and presented on the android application.

2. Universal serial bus (USB) 3.0 is used between the RF front end board and the Linux PC for sending unprocessed data. USB 3.0 features 5 Gbit/s transfer speed and has increased bandwidth as USB 3.0 is a full duplex as compared to USB 2.0 being a half-duplex. [7]

3. Wi-Fi, also known as IEEE 802.11 standard is used for communication with cell phones or PC. It will allow transmission of data from Linux machine to Android/PC. Wi-Fi operates in the 2.4 GHz

frequency range.

4. JDK, or Java Development Kit library will be used with math and hardware management functions.

5. The android application is made for android operating system Kitkat (4.4.2), but it is compatible with any device running at least android operating system Ice Cream Sandwich (4.0).

6. Linux machine runs Apache server 4.2, which is used by the android device to capture live data. The android device connects to the Linux machine server using the IP address of the computer.

7. C++ 11 was used to program the Nuand and interface all the libraries together.

## 4.3 Constraints, Alternatives, and Tradeoffs

*Constraints*

There were several constraints associated with the choice of devices. The cost of the device is not an issues since the device would be used mainly by large corporations for testing purposes. The most important constraints was the processing speed, since the data is being displayed live a fast processor was need to reduce the processing lag. The capacity of the processing units in the system need to be analyzed carefully to ensure that the signal processing load is balanced correctly in order to achieve the required rates.

*Alternatives*

In order to establish communication with the mobile device, the scanner uses Wi-Fi to connect to the established scanner. [8] Using serial connection and Bluetooth were the other alternatives to Wi-Fi. Wi-Fi offered faster speeds than Bluetooth and the wireless mobility of Bluetooth. [7][9] Therefore, despite serial communications faster data transfer speeds, Wi-Fi was chosen since it enables multiple connections to the live data. [9] Another tradeoff was using an Intel based Linux machine instead of an ARM based machine, since many of the processing libraries used were only compatible with Intel machines. ARM processors offered more portability than the x86 processors. However, dues to the libraries used Intel was the only option. Finally, in alternatives between software-defined radio platforms required, bladeRF and Hack RF had to be looked into. Eventually, bladeRF was chosen due to its higher bandwidth, sample size as well as superior power consumption as compared to Hack RF. [10]

### *Tradeoffs*

The first major tradeoff that had to be taken into consideration was that between cost and processing power. The scanner software can be run on any machine powered by x86 architecture Linux machine. Also, comparisons also had to be drawn between memory and power. The Nuand bladeRF uses Altera chips which have less RAM available per logic element than the Xilinx chips, which are used by Hack RF. However, bladeRF is better than Hack RF in terms of power consumption. It was determined that the power consumption is more important than the available RAM. [12] Final trade-off was between using iOS or Android for the mobile application. Android's openness compared to iOS's restrictions made android the better choice. Android devices are also cheaper overall than iOS devices.

## 5. Schedule, Tasks, and Milestones

The wireless scanner was designed and implemented starting over the course of the last 4 months. Appendix A lists all the tasks and milestones, the person(s) assigned to those tasks, duration and their relative difficulty level. Appendix contains the Gantt chart.

## 6. Project Demonstration

The capabilities of the Wireless Communications scanner can be demonstrated in a classroom environment through the use of an android interface on a tabloid/PC/Android/Chrome Book.

- The Nuand bladeRF is connected to a Linux machine with Intel processor with x86 architecture.
- The Linux machine requests data from the bladeRF and analyzes the raw data. The analyzed data is made available on the created apache server.
- An android phone takes in input frequency and bandwidth, which is sent to the Linux machine. Then, the phone displayed various parameters and the spectrum of the frequency and the bandwidth given.

## 7. Marketing and Cost Analysis

### <u>7.1 Marketing Analysis</u>

The target market consists primarily of individuals or corporations who wish to measure, record and graph the signal and connection quality of a cellular network. There are other RF scanning devices that can measure signal power at specific frequency ranges, but the data they provide is limited and also lack the processing power needed for multiple frequencies and protocols. One device from a company called DRTI has a similar device that can scan cellular data and display it but the interface is not as user friendly as we envision [13].

## 7.2 Cost Analysis

The cost of hardware for a prototype of this device is approximately $640. The main costs include the Nuand RF board and ODROID development boards because of their relative complexity. The DasPoint front-end interface board will be provided by DasPoint at no cost. The user must provide a mobile phone to access the information, so the user will have to handle the cost of the phone separately.

**Table 2.** Equipment Costs

| **Product** | **Quantity** | **Unit Price** | **Total Price** |
|---|---|---|---|
| Nuand RF Blade | 1 | $450 | $450 |
| Android Mobile Phone | 1 | $0 | $0 |
| Misc. Wires, Tools, Solder | 1 | $50 | $50 |
| **TOTAL COST** | | | $500 |

The development costs were determined by assuming a labor cost of $40 per hour. Fringe and overhead costs are included in the final estimate. A significant amount of time will be devoted to coding the algorithms that connect to the cellular network and communicate with the towers to get signal statistics.

**Table 3.** Development Costs

| **Project Component** | **Labor Hours** | **Labor Cost** |
|---|---|---|
| **Planning** | | |
| Group planning | 60 | $2,400 |
| Group meeting | 60 | $2,400 |
| **Hardware** | | |
| Building/Assembly | 12 | $480 |
| Testing | 12 | $480 |
| **Wireless Communications** | | |
| On-board programming | 170 | $6,800 |
| Code debugging | 170 | $6,800 |
| **Android App** | | |
| Programming & Debugging | 150 | $6,000 |
| **Demonstration** | | |
| Preparation/setup | 20 | $800 |
| **Total Labor** | 504 | |
| **TOTAL COST** | | $26,160 |

With a conservative fringe benefit cost of 30% of the total labor and overhead costs of roughly 120% of the material and labor costs, the total development cost for the prototype is estimated to be $59,065.

**Table 4.** Total Development Costs

| | |
|---|---|
| Parts | $500 |
| Labor | $26,160 |
| Fringe Benefits<br>(30% of Labor) | $7,848 |
| Subtotal | $34,898 |
| Overhead<br>(120% of Parts, Labor & Fringe) | $41,877 |
| **TOTAL COST** | $76,475 |

A production version of the complete system may include a faster, more capable front-end board with an FPGA that has more memory. This would allow for all of the computation and signal processing to be moved off of the Linux machine and onto the FPGA, effectively eliminating the need for the Linux machine. This would increase performance and reduce cost, but would take significantly more development time. The final product with more protocols implemented and other improvements would be sold to the public at a price of $10,000.

## 8. Conclusion

This project was a huge learning curve for the whole team. Before even starting the project, the team did extensive research on different network protocols and learned the algorithm for UMTS signal processing. A few changes were made to the finished product from the initial design. The front-end RF Board was not received on time due to technical issues in its development, which led the team to modify the design to accommodate its absence by adding more filters in the code and removing the GPS capabilities.  Another major design change was not using the ODroid. Initially ODroid was chosen because of the small size and powerful processor, but many of the signal processing libraries were not compatible with ARM architecture, so we used an x86 based computer running Ubuntu Linux. The last major change was using a wireless internet connection to communicate between the Android app and computer. This allowed the team to achieve the same user experience without implementing Bluetooth as we ran into trouble when trying to do so. These design changes means that we weren't able to meet our initial target for weight and size. However, our Linux code can easily be implemented in a small Linux device, which will satisfy the weight and size specifications. Additionally, there is a lot of potential for future work with our product. Algorithms for other protocols such as 4G and 4G LTE can be added to our product. A GPS signal can also be added to our device that will allow mapping of signal by location. An enclosure can also be made for our entire device, so it is easier to carry and it looks more aesthetically pleasing. Currently the scanner is able to display live spectrum and other parameters given a frequency and a bandwidth. The scanner only uses a fraction of the power that the Nuand RF Blade has to offer, which leaves the door open for further implementations and improvements to the scanner.

## 9. References

[1] Difference Between. (April 27, 2014). *Difference Between GSM and UMTS* [Online]. Available: http://www.differencebetween.net/technology/difference-between-gsm-and-umts/.

[2] Nuand Inc. (March 8, 2014). *libbladeRF C Programming Library Documentation* [Online]. Available: http://nuand.com/~jon/doc/libbladeRF-v0.14.0/index.html.

[3] Nuand Inc. (March 1, 2014). *BladeRF Getting Started* [Online]. Available: https://github.com/Nuand/bladeRF/wiki.

[4] Nuand Inc. (March 1, 2014). *BladeRF getting started installation* [Online]. Available: https://github.com/Nuand/bladeRF/wiki/Getting-Started%3A-Linux.

[5] B. Lepilleur. (March 8, 2014). *JSON C++ Programming Library* [Online]. Available: http://jsoncpp.sourceforge.net/.

[6] J. Gehring. (March 8, 2014). *GraphView library* [Online]. Available: http://www.jjoe64.com/p/graphview-library.html.

[7] Google Inc. (March 8, 2014). *Android Bluetooth library* [Online]. Available: http://developer.android.com/guide/topics/connectivity/bluetooth.html.

[8] Google Inc. (April 6, 2014). *Android HTTP library* [Online]. Available: http://developer.android.com/reference/org/apache/http/package-summary.html.

[9] Google Inc. (April 13, 2014). *Android USB library* [Online]. Available: http://developer.android.com/guide/topics/connectivity/usb/index.html.

[10] Google Inc. (April 13, 2014). *Android v7 Support Library Setup* [Online]. Available: https://developer.android.com/tools/support-library/setup.html.

[11] M. Frigo. (April 13, 2014). *FFTW Online Documentation* [Online]. Available: http://www.fftw.org/#documentation.

[12] PHP Group. (April 13, 2014). *PHP Documentation* [Online]. Available: http://www.php.net/docs.php.

[13] Apache Software Foundation. (April 13, 2014). *Apache Server* [Online]. Available: http://httpd.apache.org/.

[14] Agilent. (March 1, 2014). *IQ Modulation* [Online]. Available: http://www.home.agilent.com/upload/cmc_upload/All/IQ_Modulation.htm?cmpid=zzfindnw_iqmod&cc=US&lc=eng.

[15] Google Inc. (March 1, 2014). *JsonReader Documentation* [Online]. Available: http://developer.android.com/reference/android/util/JsonReader.html

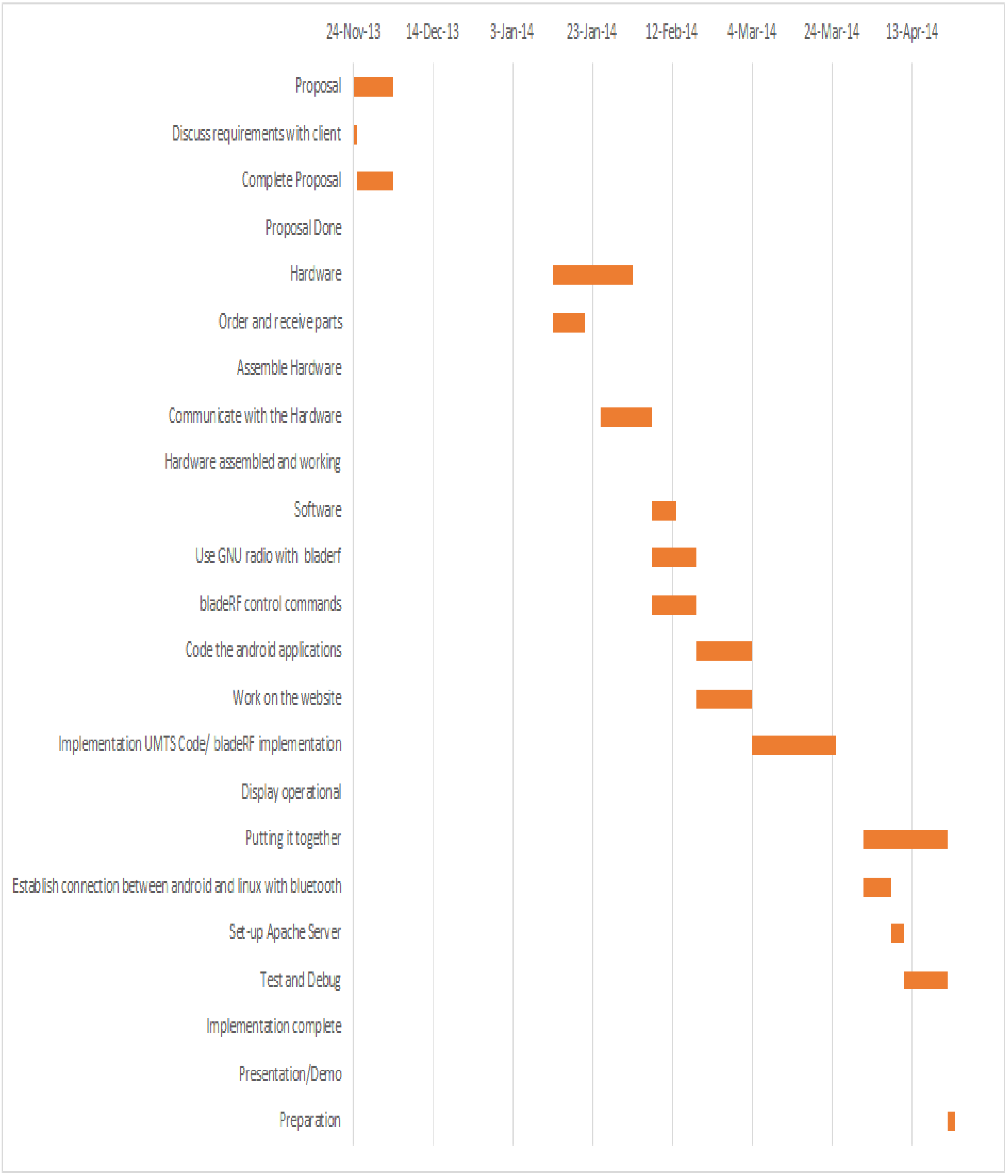
[16] Douglas Crockford. (March 1, 2014). *JSON Standard* [Online]. Available: http://www.json.org/

## Appendix A

| **Task Name** | **Duration** | **Start Date** | **Due Date** | **Responsible** | **Difficulty** |
|---|---|---|---|---|---|
| **Proposal** | 10 days | 11/24/2013 | 12/4/2013 | | |
| Discuss requirements with client | 1 day | 11/24/2013 | 11/25/2013 | Group | Low |
| Complete Proposal | 9 days | 11/25/2013 | 12/4/2013 | Group | Medium |
| Proposal Done | 0 days | 12/4/2013 | 12/4/2013 | Group | Milestone |
| **Hardware** | 20 days | 1/13/2014 | 2/7/2014 | | |
| Order and receive parts | 8 days | 1/13/2014 | 1/21/2014 | Vatsal | Low |
| Assemble Hardware | 4 days | 1/21/2024 | 1/25/2014 | Vatsal, Pooja | Medium |
| Communicate with the Hardware | 13 days | 1/25/2014 | 2/7/2014 | Team | High |
| Hardware assembled and working | 0 days | 2/7/2014 | 2/7/2014 | Vatsal, Pooja | Milestone |
| **Software** | 6 days | 2/7/2014 | | | |
| Use GNU radio with bladerf | 11 days | 2/7/2014 | 2/18/2014 | Vatsal, Prahlad, Cameron | Medium |
| bladeRF control commands | 11 days | 2/7/2014 | 2/18/2014 | Sahil, Avnish, Pooja | Medium |
| Code the android application | 14 days | 2/18/2014 | 3/4/2014 | Sahil, Prahlad, Cameron, Avnish | Medium |

| | | | | | |
|---|---|---|---|---|---|
| Work on the website | 14 days | 2/18/2014 | 3/4/2014 | Pooja, Vatsal | Medium |
| Implementation UMTS Code/ bladeRF implementation | 21 days | 3/4/2014 | 3/31/2014 | Team | Medium |
| Display operational | 0 days | 3/31/2014 | 3/31/2014 | Team | Milestone |
| **Final Build** | 21 days | 4/1/2014 | 4/1/2014 | | |
| Establish connection between android and Linux with Bluetooth | 7 days | 4/1/2014 | 4/8/2014 | Group | High |
| Set-up Apache Server | 3 days | 4/8/2014 | 4/11/2014 | Group | High |
| Test and Debug | 11 days | 4/11/2014 | 4/22/2014 | Group | High |
| Implementation comple | 0 days | 4/22/2014 | 4/22/2014 | Group | Milestone |
| **Presentation/Demo** | | 4/22/2014 | 4/22/2014 | | |
| Preparation | 2days | 4/22/2014 | 4/24/2014 | Group | Medium |

## Appendix B



24-Nov-13
14-Dec-13
3-Jan-14
23-Jan-14
12-Feb-14
4-Mar-14
24-Mar-14
13-Apr-14
Proposal
Discuss requirements with client
Complete Proposal
Proposal Done
Hardware
Order and receive parts
Assemble Hardware
Communicate with the Hardware
Hardware assembled and working
Software
Use GNU radio with bladerf
bladeRF control commands
Code the android applications
Work on the website
Implementation UMTS Code/ bladeRF implementation
Display operational
Putting it together
Establish connection between android and linux with bluetooth
Set-up Apache Server
Test and Debug
Implementation complete
Presentation/Demo
Preparation